Using User-Provided Information to Improve Internet Services

Mao Chen

Supervisor: Prof. Jaswinder Pal Singh

Co-advisor: Prof. Andrea S. LaPaugh

A DISSERTATION
PRESENTED TO THE FACULTY
OF PRINCETON UNIVERSITY
IN CANDIDACY FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE
BY THE DEPARTMENT OF
COMPUTER SCIENCE

November 2004

Abstract

Web users are not passive, but provide valuable information throughout the process of information generation, delivery and access. User-provided information is openly available in various novel web services and can often be a valuable resource for constructing further improved and enhanced services. This thesis explores the application of user-provided information in two important services, one of which is at application level while the other is at middleware level.

Our first study proposes and investigates a new reputation framework for improving rating service. Rating services allow users to harvest the collective wisdom of the broad community in making decisions. However, the difficulty with Internet ratings is that little is known about the people providing them. This thesis presents a powerful methodology that automatically computes the reputation of each online rater according to the quality and the quantity of the ratings given by the rater. This reputation information can be used to weight the ratings in aggregating multiple users' opinions on a product and to guide readers to high-quality opinions. Using data collected from real rating sites, our experiments demonstrate that our system possesses a set of important properties and has the potential to greatly enhance the effectiveness of rating service.

This thesis also proposes and investigates the utilization of user provided information in middleware design for distributed content delivery and caching. The information needs of content consumers form the key to driving content delivery over the Internet. Typically, these information needs are determined based on access patterns. This thesis explores a set of novel content placement approaches enhanced by using stated user interest through subscriptions. Our algorithms proactively deliver contents at publishing time and on demand at access time to the edge servers that are close to end-users, based on subscription and access information. We studied the algorithms' performances using a simulator and the workloads that we built to mimic the content and access dynamics of a busy news site. The results demonstrate that incorporating subscription information judiciously can substantially improve the hit rate in the local servers as compared to the access-based approaches, even when the subscription information does not reflect users' actual accesses perfectly.

Acknowledgement

First I would like to thank my thesis advisor Professor Jaswinder Pal Singh and my co-advisor Professor Andrea S. LaPaugh. For all these years, J. P. has always provided deep, acute advice on my research and taught me the important aspects of scientific research, which I take them to be: clear and consistent definition of research problems and issues, critical and thorough analysis, clear and concise communication, and above all, rigorous and persistent research. I benefited tremendously from my meetings and detailed discussions with Andrea, whose insightful advice and comments have greatly contributed to the content delivery work, as well as my other work which is beyond the scope of this thesis. I am truly grateful for the guidance, encouragement and continued support I received from J. P. and Andrea.

Professors Jaswinder Pal Singh, Andrea S. LaPaugh and Vivek Pai critically read this thesis. Their efforts and valuable comments are greatly appreciated. Vivek suggested and provided detailed comments on analyzing some new and interesting features of our content delivery algorithms, the results of which turn out to be highly valuable addition to this thesis. I would also like to thank Professor Brian Kernighan and Professor Kenneth Steiglitz for kindly agreeing to be in my committee. I enjoyed very much the vivid discussion with Ken about the application of online reputation scheme on auction sites such as eBay.

Many people have helped me initiate my research path at Princeton and they have contributed to this thesis either directly or indirectly. Professor Thomas A. Funkhouser was my academic advisor during my first year in Ph.D. program and first taught me the importance of rigorous and independent research. When I was working on online reputation system, I had

many enjoyable and valuable discussions with Dr. Dongming Jiang who gave me a lot of useful suggestions on research methodology. Dr. Stefanos Damianakis provided me with research facilities that are essential to my rating project and later work. He was also kind enough to go through in detail all my viewgraphs for my general exam. An interesting discussion with Dr. Daniel Wang instigated me to think more deeply about the practical issues of our reputation and rating system, for example, the robustness of the system against online fraud. I also had fruitful discussions with Dr. Yuqun Chen, Dr. Mingwen Ji, Ms. Fengyun Cao, and Mr. Erich Schmidt.

I would also like to thank Ms. Melissa Lawson for coordinating the whole administrative aspect of my Ph.D. program and for making it a smooth and plain process. Her helpfulness and prompt responses are always appreciated. Ms. Virginia Hogan assists my communication with J. P. when he is away, and Ms. Rebecca Davies links us graduate students to industry.

I feel very lucky that our department has a wonderful group of technical staff members who consistently put in great effort on maintaining the computing facility in the department. They are Mr. Joe Crouthamel, Mr. Steve Elgersma, Mr. Brian Jones, Mr. Paul Lawson, Mr. Chris Miller, Mr. James M. Roberts, Mr. Chris Sanchez, and Mr. Christopher J. Tengi.

Finally, I would like to thank my family. My parents are always very supportive, helping me through good and bad times and enjoying with me on my every success. I am also deeply indebted to my husband, Allan Chang. Even though he is not a computer science major, he read and edited all my papers, viewgraphs and this entire thesis, often coming up with valuable suggestions and working with me through many long nights. I am most fortunate to have his love, support, patience, sacrifice and understanding, without which I may not be able to come this far.

This work was supported by an NSF Presidential Early Career Award for Scientists and Engineers (PECASE).

To my beloved husband, Allan

and to my dear parents, Zhongli and Ruimin

Contents

Abstract . . . iii

Acknowledgement . . . v

List of figures . . . xiv

List of tables . . . xvii

List of equations . . . xviii

1 Introduction . . . 1

1.1 User Information and Internet Services . . . 1

1.2 Focus of This Thesis . . . 2

1.2.1 Reputation System in Rating Services . . . 3

1.2.2 Subscription-enhanced Content Delivery . . . 4

1.3 Organization of This Thesis . . . 5

2 Computing and Using Reputations for Internet Ratings . . . 6

2.1 Internet Rating Services . . . 6

2.2 Other Applications of Reputations and Ratings . . . 7

2.3 Related Rork . . . 8

2.3.1 Use of Reputations on Rating Sites . . . 8

2.3.2 Use of Reputations in Other Applications . . . 10

2.3.3 Related Research on Ratings 11

2.4 Key Idea of Our Approach and Validation Methodology 12

2.5 Building Reputation for Raters 13

2.5.1 Evaluating a Comment 15

2.5.1.1 Evaluating the Numeric Rating Locally 15

2.5.1.2 Evaluating Text Reviews 17

2.5.1.3 Evaluating a Comment Overall 18

2.5.2 Building the Reputation Tree 19

2.5.3 Deploying Mathematical Modules 20

2.6 Scoring Objects 21

2.6.1 Absolute Score and Confidence of Score 21

2.6.2 Use of Reputation Hierarchy 22

2.7 Experimental Results 23

2.7.1 Observations About Raw Online Ratings 23

2.7.1.1 Distribution of Mean Ratings 23

2.7.1.2 Correlation of Ratings Given by the Two Parties in Bi-directional rating 24

2.7.1.3 The Relationship Between Type of Rated Object and Distribution of Ratings 26

2.7.1.4 Summary on Characteristics of Raw Online Ratings 27

2.7.2 Experiments on Reputations 28

2.7.2.1 Properties of Reputations 29

2.7.2.2 Comparison With Reputations on Epinions 33

2.8 Summary on Reputation System for on-line Raters 37

3 Content Delivery in Publish/Subscribe Applications .39

3.1 Communication Paradigm of Publish/subscribe Systems 39

3.2 Related Work on Publish/subscribe Systems 41

3.3 Content Placement Problem in Content-intensive Publish/subscribe Services 43

3.3.1 Content Delivery for Content-intensive Pub-sub Services 44

3.3.2 Research Issues in Content Delivery for Pub-sub Applications 45

3.4 Highlights of Our Study on Subscription-enhanced Content Delivery 46

3.5 State of the Art of Content Delivery Network and Web Caching 48

3.5.1 Web Caching and Caching in CDN 49

3.5.2 Prefetching: Client-initiated Proactive Content Placement 50

3.5.3 Replication: Server-initiated Proactive Content Placement 51

3.5.3.1 Server-initiated Pushing 51

3.5.3.2 Replication Schemes in CDN 52

3.5.4 Combining Passive Caching and Proactive Content Placement 53

3.5.5 Interesting Problems in Content Delivery and Caching Algorithms 53

4 Content Delivery Algorithms Using Subscriptions 55

4.1 Architecture of Our Content Delivery Engine 55

4.2 Categorization of Algorithms 56

4.3 Algorithms for Pure Publish/subscribe Applications 58

4.3.1 Access-time Strategy Based on Access Patterns Only 58

4.3.2 Publish-time Strategy Based on Subscriptions 60

4.3.3 Publish-time and Access-time Schemes Using Subscription and Access Pattern 62

4.3.3.1 Frequency-based Approach 62

4.3.3.2 GD*-based Approaches 63

4.4 Algorithms for a More General Scenario 64

4.4.1 Our Previous Approaches Adapted for the New Scenario 65

4.4.2 Single Replacement Algorithm: Hybrid-User-GD* (HUG) 66

4.4.3 Dual Replacement Methods 67

4.4.4 Dual Caches Partitioned by Operation 68

4.4.4.1 Dual Caches Partitioned by Operation with Fixed Partition (DC-O-FP) 69

4.4.4.2 Dual Cache Partitioned by Operation with Adaptive Partition (DC-O-AP) 69

4.4.5 Dual-Caches Partitioned by Type of Access: DC-TA 72

4.5 Summary of Strategies 73

5 Simulator and Workloads for Evaluation of Algorithms 75

5.1 Architecture of System Simulator 75

5.2 Workload Generation 77

5.2.1 Generating Publishing Stream 78

5.2.1.1 Number of Pages and Publishing Time 78

5.2.1.2 Page Size and Cacheability 80

5.2.2 Generating Request Stream 80

5.2.2.1 Scaling Down the Number of Requests 81

5.2.2.2 Deciding Request Times 82

5.2.2.3 Splitting Requests by Server 84

5.2.2.4 Generating Request Traces with α = 1.0 87

5.2.3 Generating Subscription Information 88

5.2.4 Tagging Accesses as Notification-Driven or Not 90

6 Experimental Results on Subscription-enhanced Content Delivery 91

6.1 Comparing Algorithms When All Accesses are Driven by Notifications (F = 1) . . 92

6.1.1 Metrics and Experimental Setup 92

6.1.2 Influence of β 94

6.1.3 Best of Dual-Methods and Dual-Caches Partitioned by Operation 97

6.1.3.1 Analyzing the Influence of Partition in DC-FP 98

6.1.3.2 Comparing DC-O with Fixed Partition and Adaptive Partition . .98

6.1.3.3 Comparing DM and DC-O-LAP. 100

6.1.3.4 Summarizing Dual* Methods .101

6.1.4 Performance of Major Algorithms for NEWS 102

6.1.4.1 Comparison of the Approaches Under Different Cache Sizes . .102

6.1.4.2 Influence of Subscription Quality 105

6.1.4.3 Hit Ratio Versus Time 106

6.1.4.4 Workload at the Publishing Site 107

6.1.4.5 Traffic Overhead for NEWS 108

6.1.5 Performance of Major Algorithms for ALTERNATIVE Trace 112

6.1.5.1 Comparison of the Approaches Under Different Cache Sizes . .112

6.1.5.2 Influence of Subscription Quality 114

6.1.5.3 Hit Ratio Versus Time 114

6.1.5.4 Traffic Overhead for ALTERNATIVE 115

6.2 Comparing Algorithms for More General Scenario 118

6.2.1 Comparing RSG2 and HUG 119

6.2.2 Comparing Two Dual-Caches Algorithms: DC-O-LAP and DC-TA . . . 120

6.2.2.1 Comparing DC-O-LAP and DC-TA 120

6.2.2.2 Influence of Cache Partition on DC-TA 121

6.2.3 Comparing SG1, HUG and DC-O-LAP 124

6.2.4 Analyzing the Influence of Homogeneity Parameter α 126

6.2.5 Analyzing the Influence of Distribution Functions for $\{ F_{i,j} \}$ and $\{ SQ_{i,j} \}$ 128

6.3 Summary of Results 134

7 Conclusions 136

7.1 Building and Applying Reputation System in Rating Services 136

7.1.1 Contributions 137

7.1.2 Social Implications of Reputation-enhanced Rating Services 137

7.1.3 Future Work in Reputation-enhanced Rating Service 138

7.2 Using Subscription Information for Enhancing Content Delivery 140

7.2.1 Contributions 141

7.2.2 Future Work on Subscription-enhanced Content Delivery 141

7.3 Future Work on User Modeling 142

Appendix A. Properties of Function to Build *Reputation* from *GM* and *GC* 143

Appendix B. Performance of Subscription-enhanced Content Delivery Algorithms for a Workload with Very Low First Time Miss Ratio 145

Bibliography 147

Publications 156

Figures

Figure 2.1 Reputation hierarchy for one rater 14

Figure 2.2 Piecewise function for *CN* 17

Figure 2.3 Distribution of means of ratings on four ratings sites 24

Figure 2.4 Temporal trend of a rater's reputation 30

Figure 2.5 Reputation rank vs. rating experience 31

Figure 2.6 Consistency of Good and Bad Raters 32

Figure 2.7 Change in score's entropy after incorporating raters' reputations. 33

Figure 2.8 Distribution of *reputation rank* for *MTU* 34

Figure 2.9 Distribution of *CM* and *CC* of *MTU* 36

Figure 2.10 Reputation ranks of *Advisors* 36

Figure 3.1 Architecture of a publish/subscribe system 40

Figure 4.1 Deployment of content delivery engine for publish/subscribe services 56

Figure 4.2 Replacement algorithm in *GD** 60

Figure 4.3 Replacement algorithm at publish-time in *SUB* 61

Figure 4.4 Locating algorithm in DC-O-FP 69

Figure 4.5 Placing algorithm of DC-O-AP 70

Figure 4.6 Locating Algorithm of DC-O-AP 71

Figure 5.1 Architecture of the simulator 76

Figure 5.2 CDF of modification intervals for 24147 modifications 79

Figure 5.3 Publishing frequency of 30147 pages in 7 days 79

Figure 5.4 Probability of access w.r.t relative age 83

Figure 5.5 Number of requests issued at different times 84

Figure 5.6 Number of servers visiting a page in a day 85

Figure 5.7 First time miss rate at the 100 servers 86

Figure 5.8 Total number of requests and the number of distinct pages requested 86

Figure 5.9 First time miss rate at the 100 servers (α = 1.0) 87

Figure 5.10 Functions of { $F_{i,j}$ } and { $SQ_{i,j}$ } for a given global parameter F or SQ of 0.75 . .89

Figure 6.1 Hit ratios of GD*, SG1 and SG2 with different β (NEWS) 95

Figure 6.2 Hit ratios of GD*, SG1 and SG2 with different β (ALTERNATIVE) 97

Figure 6.3 Hit ratio of DC-O-FP vs. partitions (NEWS) 98

Figure 6.4 Hit ratios of DC-O with different cache sizes (NEWS) 99

Figure 6.5 Hit ratios of DC-O with different SQ at cache size = 5% (NEWS) 100

Figure 6.6 Hit ratios of three Dual* approaches 101

Figure 6.7 Hit ratios of all the methods for NEWS (*SQ* = 1, cache size from 1% to 100%) . 103

Figure 6.8 Hit ratios of all the methods for NEWS (*SQ* = 1, cache size 1%, 5%, 10%) . . . 103

Figure 6.9 Hit ratios of the algorithms with different subscription qualities 105

Figure 6.10 Average hit ratio *H* hourly (SQ = 1, capacity = 5%) 106

Figure 6.11 Workload at the publishing site 107

Figure 6.12 Traffic in number of pages (SQ = 1, capacity = 5%) 109

Figure 6.13 Traffic in bytes (SQ = 1, capacity = 5%) 110

Figure 6.14 Fraction of pushed pages that are evicted before the first request for NEWS . . .111

Figure 6.15 Hit ratios of all the methods for ALTERNATIVE (capacity from 1% to 100%) .112

Figure 6.16 Hit ratios of the approaches for ALTERNATIVE (cache size 1%, 5%, 10%) . . 113

Figure 6.17 Hit ratios with different SQ for ALTERNATIVE trace (capacity = 5%) 114

Figure 6.18 Average *H* hourly (SQ = 1, capacity = 5%) 115

Figure 6.19 Traffic in number of pages for ALTERNATIVE (SQ = 1, capacity = 5%) 116

Figure 6.20 Traffic in kilobytes for ALTERNATIVE (SQ = 1, capacity = 5%) 117

Figure 6.21 Fraction of pushed pages that are evicted before the first request for ALT 117

Figure 6.22 Performances of RSG2 and HUG 119

Figure 6.23 Comparison of performances of two Dual-Caches approaches 121

Figure 6.24 Influence of cache partition to DC-TA 123

Figure 6.25 Performances of SG1, HUG and DC-O-LAP 125

Figure 6.26 Comparing performances of algorithms for two traces 127

Figure 6.27 Comparing performances of DC-O-LAP and HUG for ALTERNATIVE 127

Figure 6.28 Performances of HUG for three distribution functions 129

Figure 6.29 Performances of DC-O-LAP for three distribution functions 131

Figure 6.30 Performances of SG1, HUG and DC-O-LAP using uniform function to model $\{F_{i,j}\}$ and $\{SQ_{i,j}\}$ 132

Figure 6.31 Performances of SG1, HUG and DC-O-LAP using stepwise function to model $\{F_{i,j}\}$ and $\{SQ_{i,j}\}$ 134

Figure A.1 $\frac{\partial \text{Reputation}}{\partial GM} - \frac{\partial \text{Reputation}}{\partial GC}$ 144

Figure B.1 Miss ratios of major algorithms when using a trace with FTM of 4.8% 146

Tables

Table 2.1 Differences between our mechanism and that of Epinions 9

Table 2.2 Four deployments of mathematical modules 21

Table 2.3 Data sets for analysis on raw online ratings 23

Table 2.4 Summary on the distributions of means of ratings on four rating sites 24

Table 2.5 Correlation of ratings given by seller and buyer in a single transaction on Yahoo . 25

Table 2.6 Sample size of three classes for objects on Epinions 26

Table 2.7 The distribution of rating profiles 26

Table 2.8 Data set for experiments on reputations 28

Table 2.9 Probability for a rater with a certain "rating experience" belonging to a certain class 31

Table 4.1 Categorization of content distribution schemes 73

Table 6.1 Review of the characteristics of important algorithms 91

Table 6.2 Relative improvements in hit ratio of different approaches over GD* (%). 113

Table 6.3 Winners for the NEWS trace at different F and SQ 125

Equations

Equation 2.1 *QN*: Quality of a numeric rating 16

Equation 2.2 *QT*: Quality of text review 18

Equation 2.3 *CT*: Confidence in QT 18

Equation 2.4 *LM*: Quality of a comment to an object 19

Equation 2.5 *CM* and *CC*: a rater's qualification for a category and corresponding confidence 19

Equation 2.6 *Reputation* of a rater for a category computed from *CM* and *CC* 20

Equation 2.7 C_S : confidence in a product score 22

Equation 2.8 *r*: Pearson's correlation coefficient between ratings from two parties 25

Equation 4.1 *V*(*p*): evaluation function of GD*. 59

Equation 4.2 *V*(*p*): evaluation function of SUB 61

Equation 4.3 *V*(*p*): evaluation function of SR 63

Equation 4.4 *f*(*p*): frequency function in SG1. 63

Equation 4.5 *f*(*p*): frequency function in SG2. 64

Equation 4.6 *f*(*p*): frequency function in HUG 67

Equation 5.1 S_i : number of server requesting a page 84

Equation 5.2 $SF_{i,j}$: the number of subscriptions matching page *i* at server *j* 88

Equation 6.1 *H*: global hit ratio at 100 servers 92

Equation 6.2 *I*: relative improvement in global hit ratio at 100 servers over GD* 118

Chapter 1

Introduction

1.1 User Information and Internet Services

In many novel Internet services, web users are not passive, but provide valuable information explicitly throughout the process of information generation, delivery and access. For example, e-commerce customers provide their feedback on products and services after transactions; the users of the notification services state their interest in web contents via subscriptions.

User information usually implies the key knowledge about the usefulness of certain contents, from the perspective of either individual users or groups of users who are clustered by location or by service. Analyzing user information is important for building novel Internet services and constructing further improved and enhanced services.

User information has been popularly used in the areas of content personalization, collaborative recommendation, etc. Specifically, many value-added services are built up based on user-provided information. With the exponential growth in web contents, an important area in the application of user-provided information is information filtering.

Rating services are powerful in helping new customers to harvest the community wisdom in e-commerce. A popular use of rating information is to reflect a product's quality based on the average rating of the product. Rating information can be also used to infer a user's preference in recommendation systems. According to the similarity in the ratings that have been given to the same products in the past, similar-minded users are grouped together, and a product is recommended to a user if the other members of the same group as the user give high ratings to the product.

A user can also form a preference profile about the information the user would like to receive from Internet services. This idea has been applied for spamming check in email services. Many notification services such as stock alerts and news feeds are based on users' subscriptions declaring their interests.

Depending on the type of service, user information can be kept, analyzed and applied at either the content provider's side or user's side. A publishing site has the complete information about its users' activities at the site. Based on the information, the site's manager can analyze the aggregated user behaviors and thus improves the services of that site. In contrast, a user log embedded in a client browser holds a single user's activities on the web. The information about a user's preferences or activities through the whole web is useful for building up personalized services at the user's side.

Besides the publishing side and the user side, an interesting component of an Internet service infrastructure for using user information is the intermediary that stands between information producers and information consumers. The intermediary can be implemented as a centralized node or an overlay network of distributed intermediaries. A distributed system is usually able in strengthening Internet services with better quality of services, scalability and availability than a centralized model, but at the expense of ease of use and maintenance.

1.2 Focus of This Thesis

This thesis discusses two independent Internet services that are enhanced by analyzing user-provided information. One is at the application level and the other at the middleware level. In addition, one focuses on publishing-side services, while the other is implemented for a group of distributed intermediary nodes between the content producers and the users.

For both problems we studied in this thesis, different approaches are proposed and compared against each other in our experiments. The results demonstrate clearly the importance of algorithm design in greatly improving solutions to each problem.

1.2.1 Reputation System in Rating Services

The first service studied in this thesis is the rating engine in e-commerce. A rating engine provides the quality information about products based on the ratings given by Internet users. Rating information is useful for helping new customers to harvest community wisdom. However, the difficulty with Internet ratings is that little is known about the people providing them. Existing services simply assume all raters are equally trustworthy and therefore they calculate a product score as a simple average of all the ratings that are given to the product.

To generate value-added information on products, raters' reputations should be taken into account in rating integration. This thesis proposes a framework to quantify a rater's reputation by analyzing the ratings given by this user and other users on all products. Given the computed reputation information, the product score is a weighted average of all ratings, with each rating being weighted by the rater's reputation. The reputation information can also serve as an indicator of the reliability of each review.

We evaluated the proposed reputation framework using data collected from real rating sites. The results show that the reputation framework built by our statistical methods has the desirable properties with regard to temporal trend, correlation between rating experience and reputation rank, greater consistency among good raters than bad raters, and the ability of reputations (and high-reputation raters) to differentiate items more finely. In addition, our automatic methodology generates results that basically match the manual judgments on raters' reliability made by site managers, while our method is able to avoid certain problems in human judgment, which is restricted by a human's ability to aggregate large and complicated data sources.

Besides Internet rating service, our reputation framework is useful in other application scenarios in which multiple parameters co-exist for generating a decision, and the quality of each parameter can be learnt from the performances of the parameter in the past.

1.2.2 Subscription-enhanced Content Delivery

The second service studied in this thesis illustrates the utilization of user-provided information in middleware design for distributed content delivery and caching. The information needs of content consumers form the key to driving content delivery over the Internet. Typically, these information needs are determined based on access patterns and pre-determinations of popular resources. This thesis studies the enhancement of content placement approaches by using stated user interest through subscriptions.

We take the "pushing to edge" philosophy to serve users' requests at the content servers that are deployed close to the end-users. As a priori knowledge about users' interests, subscription information facilitates proactive content placement to the edge servers at content generation time. Subscription information can be also used with access information to predict on the fly the future requests of a page at a content server. Our algorithms proactively deliver contents at publishing time and on demand at access time to the edge servers that are close to the users who are interested in the contents, based on subscription and access information and other factors related to a page's value.

We have studied the algorithms' performances using a simulator and workloads that we have built to mimic the content and access dynamics of a busy news site. Under various correlations between the users' pre-defined preferences and their actual behaviors, our best approaches improve the hit rate in the local servers by between 16% and 50% as compared to the baseline caching approach. When using a trace that represents more general request patterns than news requests, the relative improvement in hit rate by using stated user interest is between 33% and 133%, which is more pronounced than that for the news trace.

The results demonstrated that incorporating subscription information judiciously can substantially improve the hit rate in the local servers as compared to the access-based approaches, even when the subscription information does not reflect users' accesses perfectly.

1.3 Organization of This Thesis

The next chapter presents the algorithms to build and apply a reputation framework in Internet rating services, and discusses the performances of our algorithms. Starting from chapter 3, we investigate the application of users' pre-defined interest in building content delivery middleware system. Chapter 3 introduces the publish/subscribe communication paradigm between content providers and content consumers, and presents the challenges of content delivery in publish/subscribe applications. Chapter 4 proposes the algorithms for subscription-enhanced content delivery and caching. Chapter 5 describes the simulator and the workloads we built to mimic the content delivery service for a news site. The simulation results are demonstrated and analyzed in chapter 6. The last chapter of this thesis summarizes the major contributions of this thesis on using user-provided information for improving Internet services, and proposes some interesting directions to extend this study in the future.

Chapter 2

Computing and Using Reputations for Internet Ratings

2.1 Internet Rating Services

The Internet facilitates the circulation of opinions. The subjects span a wide range from products, services to celebrities' personalities. Anyone can give comments (in this work, a *comment* consists of a *numerical rating* and the associated *text review*) based on his/her expertise or purely on personal experience [Guernsey '00], and comments allow users to harvest the combined wisdom of the community in making decisions about products and service-providers. Users fall into two categories: *raters* who are information producers, and *readers* who are information consumers. A user may be both a rater and a reader.

Two key issues arise in this scenario. The first concerns the quality of comments provided by reviewers. The openness of the Internet makes comments very rich and valuable, but it also causes the reliability of these comments to be a key problem. This issue is especially crucial in E-Commerce where users use others' opinions as guidance before making transactions. At the same time, good raters would welcome reputation differentiation as a reward for their efforts. The reliability of raters can also be used by the site designers to better organize a site [Hafner '01]. Therefore, determining reputations for raters is a common concern for all parties.

The second issue concerns the evaluation and presentation of overall ratings for *objects* (the overall rating is often referred to as *reputation* in the literature; however, to discriminate it from the rater's reputation, it is called *score* in this study). A *score* is a summary of all numerical ratings given to an *object*. A simple average of raw ratings does not discriminate between the reliability of ratings, and thus is of limited usefulness. In addition to the reputations of raters, a good estimated product score is affected by many factors such as the number of ratings given to the object; hence it is desirable to generate a separate parameter to indicate the confidence level of each score.

The goal of this research is to build a rating aggregation system that provides the following value-added information to users:

1. The *reputations* of raters (organized as a hierarchy of categories);
2. The *scores* of objects, taking reputations of raters into account;
3. The *confidence level* of scores.

2.2 Other Applications of Reputations and Ratings

Rating service is needed in any scenario in which qualification information is important. One possible application of reputation/rating systems is to improve security and fault tolerance in Distributed Information Systems. For example, one important problem in loosely-coupled distributed information systems such as p2p networks is identifying a source's reliability. In p2p networks, each node exposes only limited information to other nodes. A reputation/rating system can be built to track the behavior of each node such as connection availability in the past and generate a rating for each node regarding its reliability in terms of availability. The replicating and searching algorithms in a p2p network can make use of this reliability rating of neighboring nodes to avoid querying unavailable nodes.

Our reputation system is also applicable to the applications in which a decision is made based on multiple parameters and the importance of each parameter can be estimated from the performance of the parameter in the past. For example, reputation information can be used to build up novel collaborative recommendation systems. Existing collaborative recommendation methods usually rely on the similarity of users' opinions. We can improve the recommendation quality by incorporating the reliability information about each user.

2.3 Related Work

The most relevant work to our study includes different reputation schemes about raters applied at rating sites. Beside that, this section also discusses some work that has been done on reputation problems in areas other than Internet ratings. Finally, this section investigates the efforts to build up Internet rating services in general.

2.3.1 Use of Reputations on Rating Sites

Available rating aggregation systems can be classified by the degree of differentiation in the reliability of ratings. In one type of system (e.g. CNET.com [CNet '04]), no differentiation in the quality of reviews is provided. Consequently, there is no help for readers to identify high-quality reviews. The second type of system supports readers' voting on the values of reviews, and tries to highlight good reviews by readers' consensus (e.g., Amazon [Amazon '00] and Deja.com, which has been acquired by eBay's [ebay] Half.com [Half]). However, no reputation about raters is aggregated on this type of site. Examples of the third type are online auction sites such as eBay.com. Here, both raters and rated objects are people participating in auctions. The "reputation" of a user can be estimated from ratings given to that user. The problem within this rating scheme is that this "reputation" is not a suitable metric for that user's qualification as a rater. For all three types, every individual rating is given the same importance in computing the overall scores of rated objects. The fourth type of rating system is the most relevant to our work on reputation system. On these sites, raters are discriminated based on their qualification, judged somehow. In calculating scores, ratings carry different weights. To our knowledge, the only site in this class is Epinions [Epinions '01].

In 2001, when our study was carried out, Epinions identified two special user groups, *Most Trusted Users* and *Advisors*. Its basic idea is to have its members identify good reviewers in one of two ways. Users may explicitly indicate trusted and distrusted users; they can also rate text reviews written by other users, and thus affect the scores of those reviews. In this scheme, a *Most Trusted User* is one who is indicated as trustworthy by many users; an *advisor* is a reviewer who is widely trusted by other users and has written many reviews with high scores. This discrimination determines the relative importance given to reviews, and in turn affects a product's score via the weight given to each review. The significant distinctions that exist between our approach and that of Epinions are listed in Table 2.1.

Table 2.1 Differences between our mechanism and that of Epinions

		Epinions	**Our methods**
Mechanism for raters' reputation	**Framework of reputation**	Binary division between good raters and others	Quantitative reputation for each rater
	Evaluation on comment	Evaluate its text review only	Evaluate both numeric and text portions, intrinsically
	Users can directly evaluate other users?	Yes	No
Evaluation of object	**Weight of a rating**	Score of its textual review	Reputation of the rater
	Need readers' explicit ratings on reviews?	Yes	No
	Confidence of score	—	Presented separately

In Epinions' method to evaluate raters, subjectivity is introduced in two ways. First, users' trustworthiness is evaluated directly by other users; which trustworthiness (and quality of ratings/reviews) should be based on an overall understanding of the whole database of ratings, but in practice a user is very likely limited in the basis for judgments of other users or their reviews. In addition, at Epinions, the evaluation of a comment is only based on its text part where bias can easily be introduced. Our method computes *reputations*, *scores*, and *confidences* implicitly from the entire database of ratings, without applying direct judgments on reputation by people. Our methodology is also different from Epinions in applying the reliability information to compute a product score. To weight a comment, Epinions requires that weights (called "ratings" then) be given explicitly to the corresponding review; our methods weigh a comment using the rater's reputation that is estimated from the quality of the user's previous comments. Therefore, our method provides more granular information in a more objective way.

The current Epinions site (2004) identifies three types of good raters according to the way they contribute to Epinions' users. *Top Reviewers* are those users who write high quality reviews in the category of their expertise. This category is in fact the *Advisor* category in the earlier classification. "*Advisors*" is now used to name the users who give high quality comments on the reviews written by other users. The third group of good raters, *Most Popular Lists*, writes reviews that are visited most. Although the categorization of helpful raters has changed, the basic methodology used to determine these special users is similar, according to the text descriptions at Epinions' site. Therefore, the above discussions on the key differences between our rating scheme and that of Epinions still hold.

2.3.2 Use of Reputations in Other Applications

Working almost at the same time as us in studying Internet ratings, Riggs and Wilensky proposed an algorithm to evaluate peer reviewers' qualifications in the context of digital scientific documents dissemination [Riggs'01]. Similar to our mechanism, Riggs and Wilensky compute reviewers' qualification automatically; and the endorsements from other reviewers on the same rated document are used as the indicator of a review's quality. For evaluating algorithms, Riggs and Wilensky also compare with Epinions.

The key algorithm of Riggs and Wilensky is a set of mutually recursive functions that calculate an item's score by weighing each rating by the qualification of the rater, where the qualification of a rater is determined based on the match of the rating to the item score. Our methodology has two major distinctions from that proposed in Riggs and Wilensky's work:

1.) Our approach incorporates the quality of text reviews in addition to ratings, while Riggs and Wilensky evaluate numeric ratings only;

2.) Our approach does not compute reviewers' qualifications and item score recursively. Riggs and Wilensky do not prove if their recursive algorithm can always converge.

2.3.3 Related Research on Ratings

An important application of online ratings is collaborative recommendation [Hill'95, Konstan'97, Resnick'94, Terveen'97]. Products rated highly by users are recommended to others with similar tastes. In collaborative recommendation systems, a product's score for a given user is determined based on the user's personal taste, and thus the calculation of objective reputation and product score is not necessary.

Zacharia and co-workers [Zacharia'99] propose a reputation system in E-communities where people rate each other. That work focuses on how reputation evolves with time based on direct ratings between people. Our method discriminates raters from rated entities, and thus is applicable in both uni-directional rating schemes for products and bi-directional rating communities for people. Compared to Zacharia's work, our reputation system is a more general and open scheme that not only incorporates users' direct ratings on other users' reviews but also analyzes the distribution of numeric ratings on each product.

Dellarocas [Dellarocas’00] proposes mechanisms to detect two types of cheating behavior. The basic idea is to detect and filter out “exceptions” in certain scenarios. Our objective is to build reputations for all raters; of course, one possible application of our reputation framework could be in detecting fake ratings from unreliable raters.

2.4 Key Idea of Our Approach and Validation Methodology

Taking advantage of the insight that reputation can be computed implicitly from ratings, we develop a general method to automatically compute reputations for raters based on the ratings they and others give to objects, and incorporate these reputations to generate value-added information about rated objects. Ultimately, a rater is reliable if the rater gives a large amount of high quality ratings. In this way, a user’s reputation is determined from the number and the quality of the ratings the user gives.

People’s expertise usually varies across knowledge domains, hence a rater’s reputation is broken down by category of rated objects. A rater’s reputation for a category combines the rater’s qualification for rating objects in that category and the confidence in that qualification. A rater’s qualification for a category is computed based on the qualities of all the opinions given by the rater in the category, while the confidence in the computed qualification for the user and the category is measured from the number of ratings given by the user in that category.

In many on-line rating systems, people are allowed to submit not only the numeric ratings but also text reviews on products. A user’s opinion on a given product may consist of a rating and a review. Our algorithm evaluates the two portions separately, and then computes the opinion’s quality by aggregating the two results. The quality of a numeric rating is determined from the consistency of the rating to other ratings given to the same product. The evaluation of a text review is based on a consensus model on readers' voting. When many readers indicate a review is helpful, the review's quality is expected to be high.

Taking advantage of the computed reputations of raters, we use reputation-weighted ratings as the overall score for each object. To capture other factors affecting the reliability of a score such as the number of ratings given to the object or the reputations of the raters evaluating the object, we compute a separate parameter to indicate the confidence level of each score.

The quantitative experiments in this study are carried out with two goals. One is to analyze the characteristics of two major types of online ratings: bi-directional ratings for online auctions and uni-directional ratings for various objects. Another goal is to analyze our mechanisms for computing reputation-enhanced ratings from two perspectives. First, we demonstrate that our reputation framework and methods indeed satisfy a set of desired properties. Second, our methods for computing reputations are compared to Epinions. We find that the raters with high computed reputation by our methods correspond well to the good raters chosen by site managers and users on Epinions. However, our methods provide additional, important, more granular information and thus are more discriminating than human judgments.

2.5 Building Reputation for Raters

We first discuss how we compute reputations for raters. In the next section, we will discuss how these reputations are used to compute scores of rated objects.

Two properties are assumed to be associated with a rater's reputation in our work:

(1) It is a dynamic random variable, reflecting uncertainty.

(2) It is classified based on expertise category.

The uncertainty of reputation is partly due to variability of every person's expertise with time, and partly because of updates in direct or indirect knowledge our system has about raters over time. The second property is encoded in our bottom-up approach in building a reputation hierarchy for each user from leaf categories to the root category.

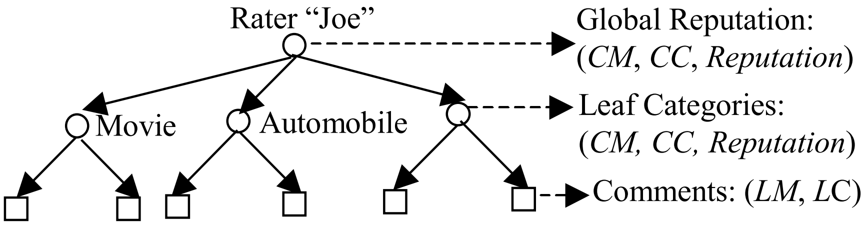


Figure 2.1 Reputation hierarchy for one rater

Figure 2.1 illustrates a reputation tree for a rater based on the classification of objects into categories. Rectangles denote comments that include raw ratings and text reviews, and circles refer to categories. The categories on the lowest level are called *leaf categories*. The reputation information in the non-leaf categories is aggregated from information in their children.

The procedure to build a reputation tree consists of three stages. First, every comment that consists of a numeric rating and a textual review is evaluated using information regarding that comment and other comments on the same object. The *local match* (*LM*) denotes the resulting quality of one comment. The *local confidence* (*LC*) is the confidence in *LM*. Second, the rater's reputation in the leaf categories is calculated based on $\{(LM, LC)\}$ for all comments by that rater in that category. The overall quality of all comments by that rater in a category is computed and referred to as *category match* (*CM*), and the confidence in *CM* is computed and referred to as *category confidence* (*CC*). A rater's *reputation* in a category is built by combining the rater's *CM* and *CC* for that category. Finally, information regarding reputation in all non-leaf categories is generated. The ranges of all variables mentioned above are [0, 1].

2.5.1 EVALUATING A COMMENT

As the first step in the three stages described above, evaluating an individual comment can be split into two parts: evaluation of its numeric rating and of its textual review. The quality of a comment as a whole is based on these two evaluations. The formulae in this section can be easily extended to any rating scales, though a rating scale of 5 is assumed in this discussion.

2.5.1.1 Evaluating the Numeric Rating Locally

The only information we use to evaluate an individual numeric rating is the frequency distribution of all ratings given to the same object.

First, raters are grouped according to the rating values they give to the object. Every rater is deemed to give the highest endorsement to other raters in that rater's group. The endorsement of raters to other groups is a function of the discrepancy between the rating levels of the groups.

The endorsement of one group to another group could simply be a function of difference of two ratings. However, the semantics of rating levels provide the hints to define endorsements in a more meaningful way. For example, the discrepancy between "5" (excellent) and "3" (neutral) is smaller than that between "4" (good) and "2" (bad) intrinsically, though the numerical gaps between the pairs are same. The following endorsement matrices (before normalization) are examples of the above two endorsement measurement schemes. E_1 is simply a linear function of the difference of two rating levels, while E_2 factors in the meaning of rating levels. For example, according to E_1 the endorsement of a rating group to any other group that is two rating levels away is 0.6. Using E_2, in contrast, the endorsement of the rating group at "5" to the group at "3" is 0.2, but that of "4" to "2" is 0 because they are fundamentally opposite opinions. In our implementation we normalize the matrix E so that the total endorsement of one group to all groups including itself is 1.

$$E_1 = \begin{bmatrix} 1 & 0.8 & 0.6 & 0.4 & 0.2 \\ 0.8 & 1 & 0.8 & 0.6 & 0.4 \\ 0.6 & 0.8 & 1 & 0.8 & 0.6 \\ 0.4 & 0.6 & 0.8 & 1 & 0.8 \\ 0.2 & 0.4 & 0.6 & 0.8 & 1 \end{bmatrix} \qquad E_2 = \begin{bmatrix} 1 & 0.8 & 0.2 & 0 & 0 \\ 0.7 & 1 & 0.3 & 0 & 0 \\ 0.2 & 0.3 & 1 & 0.3 & 0.2 \\ 0 & 0 & 0.3 & 1 & 0.7 \\ 0 & 0 & 0.2 & 0.8 & 1 \end{bmatrix}$$

Endorsements based only on difference of ratings

Endorsements taking into account semantics of rating levels

The quality of a rating at the rating level *i*, denoted as QN_i, is the sum of endorsements from all groups/rating levels, weighted by the *importance* of each endorsement. This importance is the *QN* of the endorsing group. To make the computation converge, the relative group size is used as the *a priori importance*. $\{QN_i\}$ is computed by solving Equation 2.1.

$$QN_i = \sum_{\substack{j=1 \\ j \neq i}}^{5} QN_j \bullet E_{j,i} + \frac{N_i}{\sum_{j=1}^{5} N_j} \bullet E_{i,i} \qquad \text{—Equation 2.1}$$

Where
$E_{j,i}$: the endorsement of level j to level i
$E_{i,i}$: a constant $\in [0,1]$, for any i
N_i : the total number of raters giving a rating of i

Our method to analyze the distribution of ratings is similar to the method of *weight propagation* or *transfer of endorsement* that is used to rank pages in some search engines [Brin'98, Kleinberg'99]. Each rating group in our problem can be viewed as a page node in the analysis on page ranks. Similar to Brin and Page's work, our approaches do not distinguish an authority from a hub, or in other words, an authority node that represents a high quality rating is also a hub that endorses other good ratings, which is different from the analysis on pages' importance in Kleinberg's work.

In practice, "rating scarcity" is a common problem for rating aggregation systems. Measuring the confidence of a result is important but is often ignored. The confidence of $\{QN_i\}$ is determined by the sample size *N* (the total number of ratings given to an object). Based on the discussion about the confidence of index numbers in [Arkin'70], a piecewise function as shown in Figure 2.2 is designed to compute a confidence level (*CN*) for *QN*. The function grows slowly when *N* is smaller than 20; it grows faster when *N* is between 20 and 50; after that *CN* still grows but gradually; and when *N* is beyond 200, increasing *N* has no effect on *CN*.

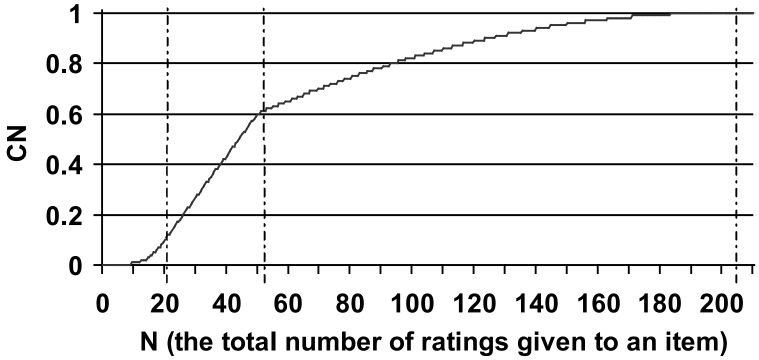


Figure 2.2 Piecewise function for CN

The method discussed in this section is based on the assumption that the quality of objects can be evaluated somehow by majority rule. For objects outside this assumption, the information consumers can decide to what degree they accept the raw ratings and other value-added information generated from raw ratings.

2.5.1.2 Evaluating Text Reviews

For sites that support readers' voting, the quality of a text review can be estimated from readers' explicit ratings of that review. In this computation, every reader's rating is weighted based on the current reputation of the reader. Thus, a user can be an information producer and an information consumer for the same object (text review). The quality of a text review and the corresponding confidence are denoted as *QT* and *CT*.

When computing *QT*, Equation 2.2 factors the reputations of the readers in, assuming a good rater on products also has good judgements on reviews. In Equation 2.2, the parameter *M* is used to normalize the resulting *QT*.

$$QT = \frac{\sum_{i=1}^{N}(r_i \times R_i)}{M \times \sum_{i=1}^{N} R_i} \qquad \text{—Equation 2.2}$$

Where

N : the total number of ratings to the review

r_i : the ith rating on the review

R_i : the reputation of the user who gives r_i

M : the rating scales used on text reviews, this value is 4 on Epinions

The confidence in *QT* is determined by readers' qualifications as rater as follows:

$$\text{If } \sum_{i=1}^{N} R_i < C \qquad \text{— Equation 2.3}$$

$$\text{Then} \quad CT = \frac{\sum_{i=1}^{N} R_i}{C}$$

$$\text{Else} \quad CT = 1$$

Where

R_i : the reputation of the ith rater

C : the threshold for the number of reviewers with reputation 1 to gain a full confidence in the overall rating on a text review. 50 is used in this study.

2.5.1.3 *Evaluating a Comment Overall*

Equation 2.4 shows how we put the computed evaluations of the two parts of a comment together to generate *LM*, the quality of a comment, and *LC*, the confidence in *LM*. The resulting local confidence *LC* is based on discussions on the standard error of arithmetic mean [Arkin'70]. Because we do not want a linear correlation between the confidence and the standard error, *LM* is equal to the difference of 1 and the square root of the standard error as shown in Equation 2.4. Not trivially, the evaluation on either portion of a comment can be used to estimate the overall quality of a comment, if the other portion is unavailable.

$$LM = \frac{QN \times CN + QT \times CT}{CN + CT} \qquad \text{—Equation 2.4}$$

$$LC = 1 - \sqrt{E}$$

Where

$$E = \sqrt{\frac{CN^2 \times (1 - CN)^4 + CT^2 \times (1 - CT)^4}{(CN + CT)^2}}$$

2.5.2 Building the Reputation Tree

As mentioned earlier, a given rater's reputation is organized as a tree based on the category structure of objects. Each category node in the tree is associated with three values: *Category Match* or *CM* is the estimate of the rater's qualification for that category; *Category Confidence* or *CC* measures the confidence in *CM*; *Reputation* is a quantitative summary about the rater's qualification in that category based on *CM* and *CC*. *CM* and *CC* for a category are determined from the *matches* and the corresponding *confidences* of its children using Equation 2.5.

$$CM = \frac{\sum_{i=1}^{N} (M_i \times C_i)}{\sum_{i=1}^{N} C_i} \qquad \text{—Equation 2.5}$$

$$CC = 1 - \sqrt[4]{\frac{\sum_{i=1}^{N} \left(C_i^2 \times (1 - C_i)^4\right)}{\left(\sum_{i=1}^{N} C_i\right)^2}}$$

Where
N : the total number of items rated by the rater under the category
M_i : the match of the rater in the ith subcategory or LM_i for the ith item in a leaf category
C_i : the confidence in M_i

There are multiple options for computing the *reputation* of a rater in a category from *CM* and *CC* for that category. The basic requirements of a reputation function include:

1.) It increases with *CM* or *CC* when the other variable is fixed.

2.) It is 0 if either *CM* or *CC* is 0, which implies that either all comments given by the rater are very bad, or the relevant information is insufficient to generate any reputation.

3.) It is 1 only when both *CM* and *CC* are 1.

4.) It is exactly the same as *CM* if *CC* is 1.

Some other properties as follows are also desirable:

1.) The bigger *CC*, the faster reputation grows with *CM*.

2.) The bigger *CM*, the faster reputation grows with *CC*.

3.) *CM* is dominant when *CC* is large but *CM* is small.

4.) *CC* is dominant when *CM* is large but *CC* is small.

A formula satisfying all of the above basic and supplementary requirements is given by Equation 2.6. The proof that the following reputation function satisfies all the properties is given in appendix A.

$$\text{Reputation} = (CM + 1)^{CC} - 1 \qquad \text{—Equation 2.6}$$

2.5.3 Deploying Mathematical Modules

The modularity of our mathematical model allows modules to be loaded and combined in a flexible way to build reputation. Our reputation framework consists of several pluggable modules. In particular, either part of a comment (numerical or textual) or both parts can be used in comment evaluation as desired; for reputation organization, choices can be made between a flat structure and a category-based hierarchical structure.

In this work, four mechanisms to build a rater's reputation are discussed, as defined in table 2.2. Among the four models, the least computationally expensive model is FLAT model that ignores the categorization of rated objects and the qualities of text reviews. The model that involves all the modules introduced above is TREE&TEXT. In general, different deployments make different trade-offs between the computational complexity and the quality of results. An experimental comparison of these models will be shown in Section 2.6.2.

Table 2.2 Four deployments of mathematical modules

		Organization of Reputation	
		Flat	**Tree**
Evaluate text reviews too?	**No**	FLAT	TREE
	Yes	FLAT&TEXT	TREE&TEXT

2.6 Scoring Objects

A direct application of the reputation framework built using the approaches in section 2.5 is generating product score. Raters' reputations can be used as weights in aggregating multiple users' opinions on an object, and in estimating the reliability of a score.

2.6.1 Absolute Score and Confidence of Score

For information consumers, the central concern is the evaluation of objects, denoted as *score*. The *score* of an object is calculated as the weighted average of all its ratings, with the weight of a rating being the rater's reputation. This is more meaningful than a simple average of raw ratings as used in many existing rating systems.

If only few, not very reliable people have rated an object, one cannot be very confident in the resulting score. Inspired by the statistical method to estimate the standard error of arithmetic mean, the confidence of a score is determined by at least three factors as follows:

1.) The total number of ratings given to the object

2.) Reputations of raters who give these ratings

3.) Degree of consistency in ratings

Equation 2.7 is the formula we used to calculate the confidence of a score, denoted as C_S. C_S of a product's score increases with the total number of ratings given to the product, the reputations of raters who give the ratings, and the degree of consistency in the ratings.

$$\text{If } \sum_{j=1}^{N} R_j \le 1 \text{ then } C_S = 0$$

Else

$$C_S = 1 - \sqrt{E}$$

$$E = \sqrt{\frac{\dfrac{\sum_{j=1}^{N}\left(r_j^{\,2} \times R_j\right)}{\sum_{j=1}^{N} R_j} - S^2}{\sum_{j=1}^{N} R_j - 1}} \qquad \text{—Equation 2.7}$$

Where

N : the number of ratings given to the rated object

r_j : the jth rating to the item

R_j : the reputation of the rater who gives r_j

S : the overall score of the item

2.6.2 Use of Reputation Hierarchy

When computing a score for an object, the weight of a rating may be chosen to be the rater's reputation for any category on the path from the root of that rater's reputation tree to the leaf category including the object under evaluation. Using a user's reputation for a given category as the weight of the ratings that the user gives to the products in that category emphasizes the rater's expertise in the given category.

However, when there is lack of information about the raters at finer granularity, which is reflected by small *Category Confidence* (*CC*), it is desirable to apply reputations for a coarser-grained category in computing product score. Therefore, every object can be given a set of scores based on raters' reputations at different category levels.

2.7 Experimental Results

Our experiments are divided into two sections. The first section is a set of observations on raw online ratings. The second section is an analysis of our methods to build reputations and scores.

2.7.1 Observations About Raw Online Ratings

There are several ways to classify online ratings. Ratings can be discriminated as uni-directional or bi-directional ratings, or be classified as ratings on people or ratings on products. For example, ratings on online auctions are usually bi-directional and the rated objects are people involved in transactions. Many shopping sites or rating intermediary sites like Epinions collect uni-directional ratings for a wide range of objects.

Two data sets are used in this section. The first includes bi-directional ratings of people, collected from three auction sites: eBay [ebay], Amazon [amazon], and Yahoo [Yahoo]. The second data set includes uni-directional ratings of different objects on Epinions [Epinions]. All the data was collected in the spring and summer of 2000.

Table 2.3 Data sets for analysis on raw online ratings

	Auction Sites			**Epinions**
	Amazon	**Yahoo**	**Ebay**	
Number of objects with more than 10 ratings	16,377	59,293	212,285	5,974

2.7.1.1 Distribution of Mean Ratings

Figure 2.3 shows the distribution of the mean ratings for rated objects. Most users on auction sites, either sellers or buyers, have mean ratings at the highest rating level. Compared to auction ratings, the distribution of mean ratings on Epinions spreads over a wider range on the rating scale.

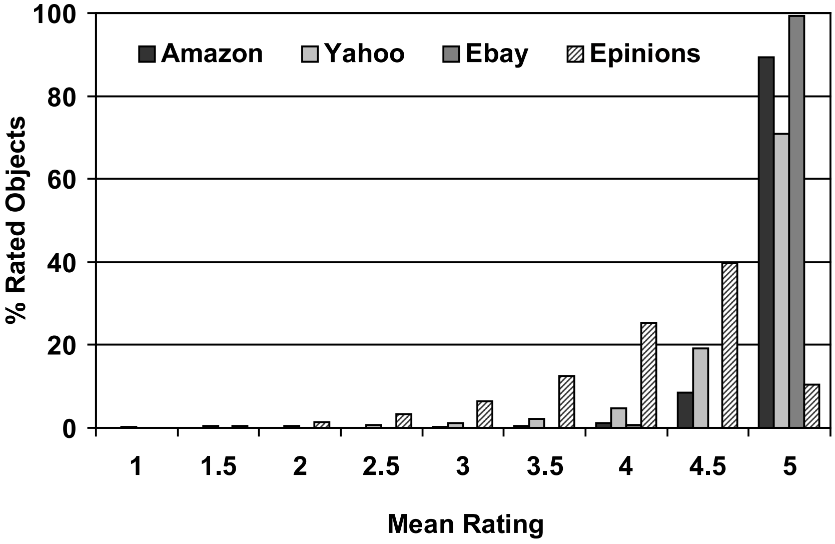


Figure 2.3 Distribution of means of ratings on four rating sites (eBay uses rating scale of 3, so the ratings from eBay are normalized to scales of 5 in our experiments)

Table 2.4 gives more details about the distributions in Figure 2.3.

Table 2.4 Summary on the distributions of means of ratings on four rating sites

Sites % Objects for which:	Auction sites			Epinions
	Amazon	Yahoo	Ebay	
Mean of ratings is above 4	98	90	99.36	51
Mean of ratings is below 3	0.24	1.6	0	5.3

2.7.1.2 Correlation of Ratings Given by the Two Parties in Bi-directional Rating

There are several possible reasons why many people have high scores on auction sites:

1. People do well in online transactions.
2. People tend to give high ratings as long as the other party pays or sends the goods.
3. Some sellers rig the system to bolster their ratings.
4. *Bi-directional rating schemes lead users to give high ratings to others in the hope of getting high ratings in return.*

The first two reasons are beyond the scope of this study. The cheating behavior in the third case is not enough to explain the ubiquity of high scores on auction sites. In this work we focus on the fourth possibility.

Based on user IDs, 1,056,368 transactions were extracted from ratings given to 295,930 users on Yahoo. The ratings given by the seller and buyer in a single transaction are paired together. Table 2.5 presents the summary of raw results.

Table 2.5 Correlation of ratings given by seller and buyer in a single transaction on Yahoo

Cases		**% Transactions**
The ratings that two parties give to each other are	exactly same	91
	both high ratings (≥4)	95
When one party gives a low rating (≤2), the other party gives:	a low rating (≤2)	56
	a high rating (≥4)	29
	a neutral rating (=3)	15

Ignoring the differences between 4 (good) and 5 (excellent), and 2 (bad) and 1 (very bad), the Pearson's correlation coefficient between ratings from two parties *r* (defined in Equation 2.8) is about 0.72, which shows a high positive correlation. Based on observations in Figure 2.3 and Table 2.4, we believe this positive correlation is stronger on Amazon and eBay than Yahoo. The results support our hypothesis that a bi-directional rating scheme leads to high ratings in online auctions.

$$r = \frac{\sum XY - \frac{\sum X \sum Y}{N}}{\sqrt{\left(\sum X^2 - \frac{\left(\sum X\right)^2}{N}\right)\left(\sum Y^2 - \frac{\left(\sum Y\right)^2}{N}\right)}} \qquad \text{—Equation 2.8}$$

Where

N : the number of transactions

X : the rating vector of sellers

Y : the rating vector of buyers

2.7.1.3 The Relationship Between Type of Rated Object and Distribution of Ratings

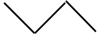
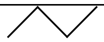
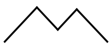
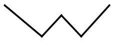
For uni-directional ratings on Epinions, the rated objects span a wide range of products, services, and other topics such as travel destinations or celebrities. To investigate the relation between types of rated objects and distributions of ratings, the objects on Epinions are classified according to their nature in terms of the difficulty of being evaluated objectively. We defined three classes for objects: "*Objective*"—the quality can be determined by fairly objective standards (e.g. electronic products); "*Subjective*"—the evaluation is strongly based on personal taste (e.g. travel destinations); "*Medium*"—the evaluation is based on a mixture of objective standards and personal expectations (e.g. services).

Table 2.6 Sample size of three classes for objects on Epinions

	Objective	***Medium***	***Subjective***
# Objects with more than 10 ratings	1926	2198	1850

The three classes were first compared in the distributions of their mean ratings by experiments discussed in sections 2.7.1.1. The distributions for the three classes show similar properties as the whole data set before classification; namely, ratings of many objects are high. The three classes of objects are then compared in the shape of rating profiles for a single object. We classified all possible rating profiles into 10 groups, as described in Table 2.7, and counted the relative occurrence of each shape for each class.

Table 2.7 The distribution of rating profiles

	***Obj.*(%)**	***Med.*(%)**	***Sub.*(%)**
Monotonically Increasing	33.2	34.1	39.1
Bi-modal ("U" shaped)	24.8	20.7	24.0
Uniform Distribution	12.3	10.1	10.3
Bell-shaped	7.6	12.2	11.0
	9.1	9.0	5.2
	5.9	5.5	4.4
 ("M" shaped)	3.1	3.6	1.9
Monotonically Decreasing	2.7	2.8	2.2
 ("W" shaped)	1.0	1.4	1.0
A Single Rating Group	0.3	0.6	0.8

The distributions described in Table 2.7 can be interpreted as follows. Uniform distribution represents high degree of dispersion in users' opinions to a same object, or no dominating opinion about the rated object; "W" shaped distribution can be viewed as an extension of strictly uniform distribution. Bi-modal or "U" shaped distribution demonstrates that two opposite opinions dominate users' ratings on a same object; "M" shaped curve also reflects two dominating opposite ratings but with a lower degree of discrepancy between the two ratings; ╲╱╲ and ╱╲╱ curves exhibit the mildest discrepancy when there are two dominating opinions of an object. Monotonically changing curves or bell-shaped ones demonstrate that users' opinions converge to a common rating, though the degree of the concentration of users' opinions is not as high as represented by a single rating group.

Some interesting observations are obtained from Table 2.7:

1) The distributions of possible rating profiles are similar for all three classes;
2) The possible rating profile for a single object may be any of the 10 types of shapes;
3) Monotonically increasing and "U" shaped distributions are dominating cases after which are uniform distribution and bell-shapes.

2.7.1.4 Summary on Characteristics of Raw Online Ratings

The above analysis results on raw ratings have several important implications in designing our rating aggregation methods:

1) It is difficult to perform a meaningful or non-trivial evaluation on raters and rated objects from bi-directional ratings in online auctions, which is due to the ubiquity of high ratings. We therefore do not use the auction ratings in the performance analysis on our reputation scheme, described in the next section;

2) Given the lack of correlation between the distribution and the subjectivity of ratings, a general method can be applied to evaluate raters and objects for uni-directional ratings, regardless of the type of rated object in terms of expected subjectivity of ratings;

3) Rating analysis on an object needs to take into account all possible distributions.

2.7.2 Experiments on Reputations

The kernel of our rating aggregation system involves building reputations for raters. The data used in this section are the ratings from Epinions during the summer of 2000. The structure of the reputation tree for a rater is based on the classification of categories on Epinions. In our experiments, the depth of a reputation tree is 2 and all leaf categories are at the same level.

Table 2.8 Data set for experiments on reputations

Number of Objects	Number of Raters	Number of Categories at level 1	Number of Leaf categories
50,725	59,986	14	80

Equation 2.6 generates a real number between 0 and 1 as a rater's reputation for a category. To illustrate a rater's qualification as compared to the other raters, we convert each raw reputation into a *reputation rank*. Our algorithm to generate reputation rank is based on the Square-Error clustering method. The total number of rank clusters is set as 10. In a rater's reputation hierarchy, each reputation value is associated with a reputation rank in the corresponding category.

While there is no gold-standard metric to compare with for evaluating raters' reputations, we tested our methods from two perspectives. First, the rater's reputations as computed by our system are analyzed against some desirable properties of reputations. Second, for raters who are (independently) distinguished as being trusted or advisors on Epinions, the reputations computed by our mechanisms are presented and analyzed. The second approach is also adopted in Riggs' work to evaluate their algorithms for evaluating qualifications of on-line reviewers on scientific documents [Riggs' 01].

2.7.2.1 Properties of Reputations

Some important, desired properties of a system for computing rater's reputation include:

1.) Reputation depends on both quality and quantity of all ratings given by a rater.

2.) Rating experience should not directly determine reputation rank for a rater, though reputation generally grows with rating experience for an individual rater.

3.) Good raters are more consistent in their ratings for the same object than bad raters.

4.) Good raters have greater ability to be discriminating among objects in the same category.

Criterion 1 is built-in to our method for computing a rater's reputation. The experiments in this section test our system against the other three properties. The model used in this section is TREE (see Table 2.2). Raters in each leaf category are classified into three groups: *Good Raters,* whose *reputation rank* is not lower than 7, *Bad Raters* (below 4), and *Others* (between 4 and 6, inclusive).

(1) Rater Reputation as Experience Increases

The goal of this experiment is to investigate the evolution of a rater's reputation as their rating experience increases. In this experiment, raters' reputations at leaf category level are analyzed, and a rater's *rating experience* in a category is measured as the total number of ratings the rater gives in that category.

Within every leaf category, every rater's rating history is sorted according to the generation time of the ratings. Putting emphasis on users who have long enough rating history, this analysis includes only raters who give at least 50 ratings in the category. To avoid the "cold start" problem, after a rater gives 10 ratings in a category, the reputation for that rater is calculated and plotted on a curve. The curves are aggregated into the three groups according to raters' final reputation rank for that category. In Figure 2.4, the y-axis is the average reputation for raters in the same group at each time point.

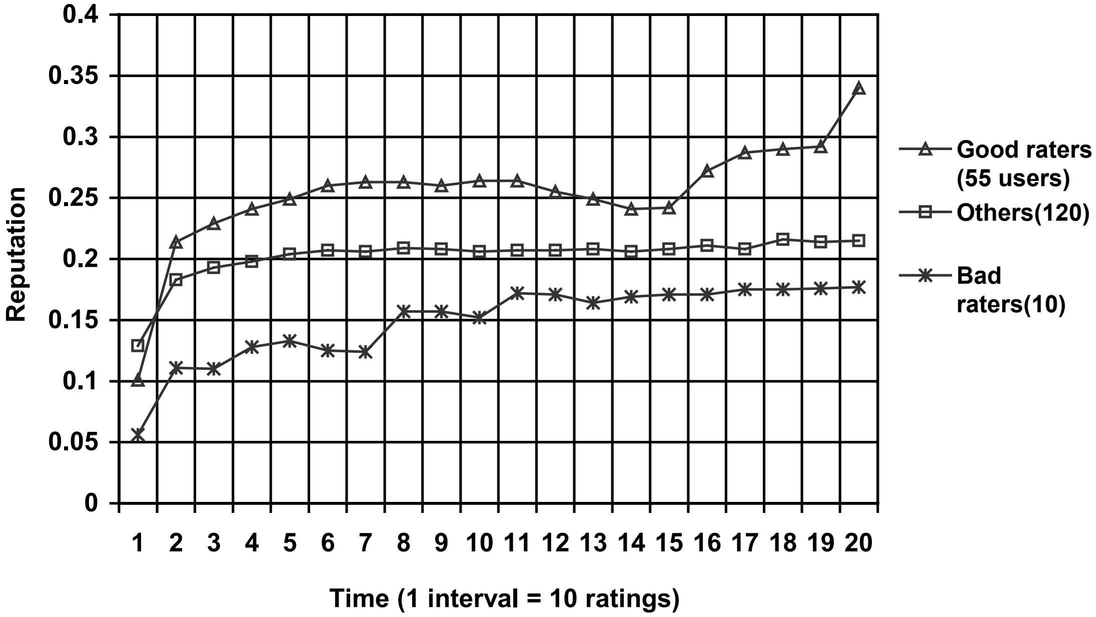


Figure 2.4 Temporal trend of a rater's reputation

In general, a rater's reputation increases with rating experience. However, the three groups exhibit different properties in reputation's evolvement. There are two stages when the reputation of *Good Raters* increases at the highest speed: the "warm-up" stage (while giving the first 70 ratings) and the "maturing" stage (after giving 140 ratings). For *Bad Raters*, the reputation grows in a step-wise manner. *Others* gain their medium reputation ranks mainly from the reputation accumulated in the "warm-up" stage. For *Bad Raters* and *Others*, low starting point and inability to provide high quality comments cause their lower reputation than *Good Raters*, respectively.

The next experiment shows that reputation cannot be determined from rating experience directly. First, Table 2.9 shows the lack of functional dependency between the "experience" and the class a rater belongs to. Each cell in Table 2.9 shows the probability for a user whose rating experience falls into a given range belonging to a certain rater class.

Table 2.9 Probability for a rater with a certain "rating experience" belonging to a certain class

Total number of ratings	[0,50]	[50,100]	[100,150]	[150,200]
Good Raters	38%	29%	46%	33%
Others	41%	45%	54%	33%
Bad Raters	21%	26%	0%	33%

In Figure 2.5, each point represents the average reputation rank of the raters of a rater class with a given rating experience. Within each rater class, the lack of correlation between rating experience and reputation is quite clear.

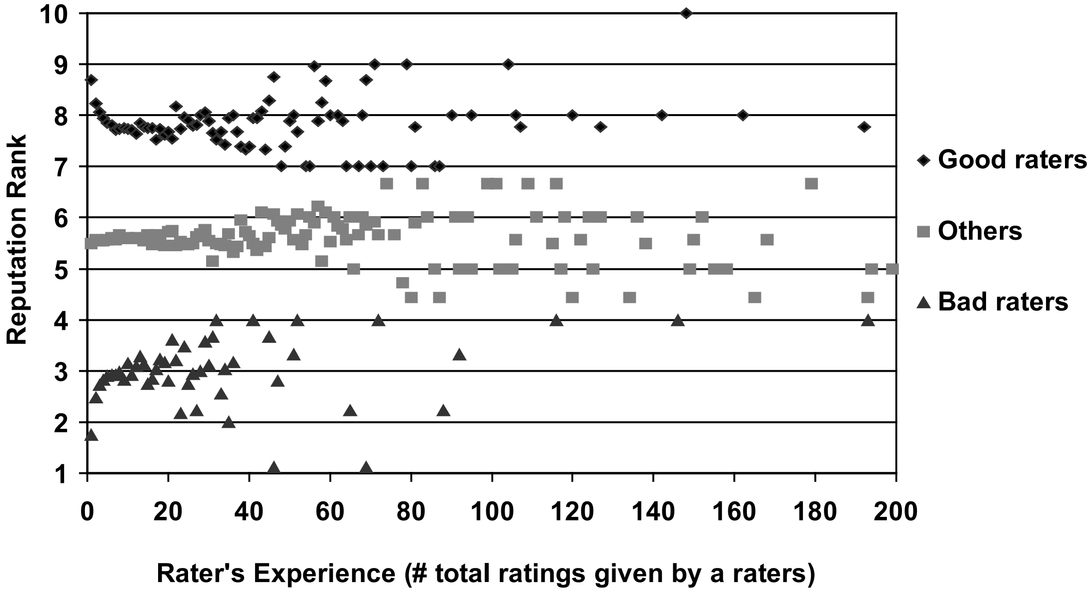


Figure 2.5 Reputation rank vs. rating experience

The results that reputation can not be determined from rating experience directly in our reputation system shows that our method effectively incorporates other important factors such as the quality of comments (including numeric ratings and textual reviews) in addition to the amount of ratings given by a rater.

(2) Consistency of *Good Raters* and *Bad Raters*

In this experiment, *Good Raters* and *Bad Raters*, defined as before, are compared in the consistency of ratings given by members of each group to the same object. Only items rated by at least 5 *Good Raters* and 5 *Bad Raters* are sampled, and only raters who have given more than 5 ratings in the corresponding leaf category are included. The inconsistency of ratings is measured using the standard deviation of the ratings given by a group of raters to an object.

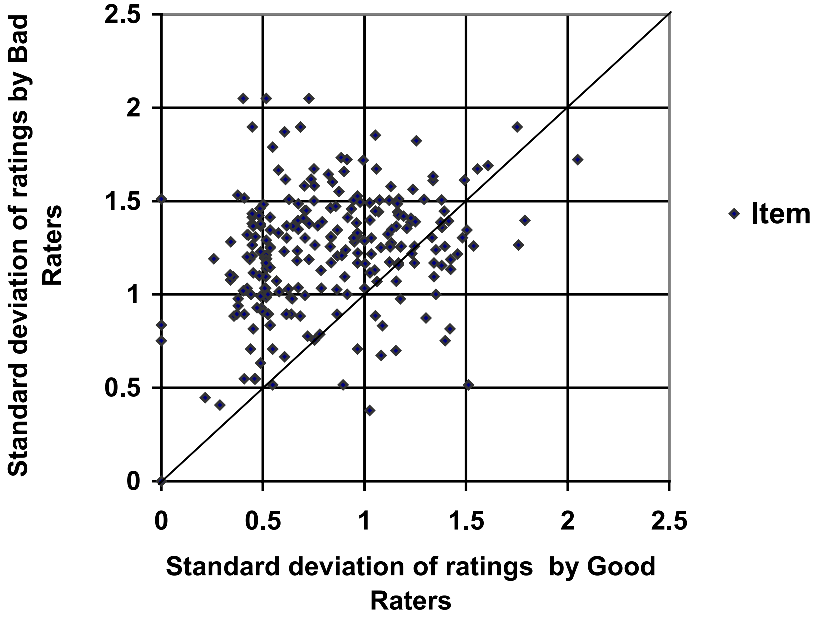


Figure 2.6 Consistency of Good and Bad Raters

In Figure 2.6, a point represents one object. The x-axis and y-axis give the standard deviation of ratings given to an object by *Good Raters* and *Bad Raters*, respectively. For more than 85% of the 246 objects, the standard deviation of *Good Raters'* ratings is much smaller than that of *Bad Raters*' ratings. The average of standard deviation of *Good Raters*' ratings is 0.83, while this value is 1.23 for *Bad Raters*. So *Good Raters* as computed by our methods are indeed more consistent in their ratings than *Bad Raters*.

(3) Ability to Discriminate Among Items Within a Category

Intuitively, good raters should be more discriminating, and are therefore more valuable as raters. We define the *entropy* of product scores in a category as the standard deviation of scores for all objects in that category. The change in entropy resulted from applying reputations in calculation of scores is measured. To keep the confidence level, this experiment considers only categories that include more than 30 rated objects, yielding all 14 categories on the first level and 59 of 80 leaf categories analyzed.

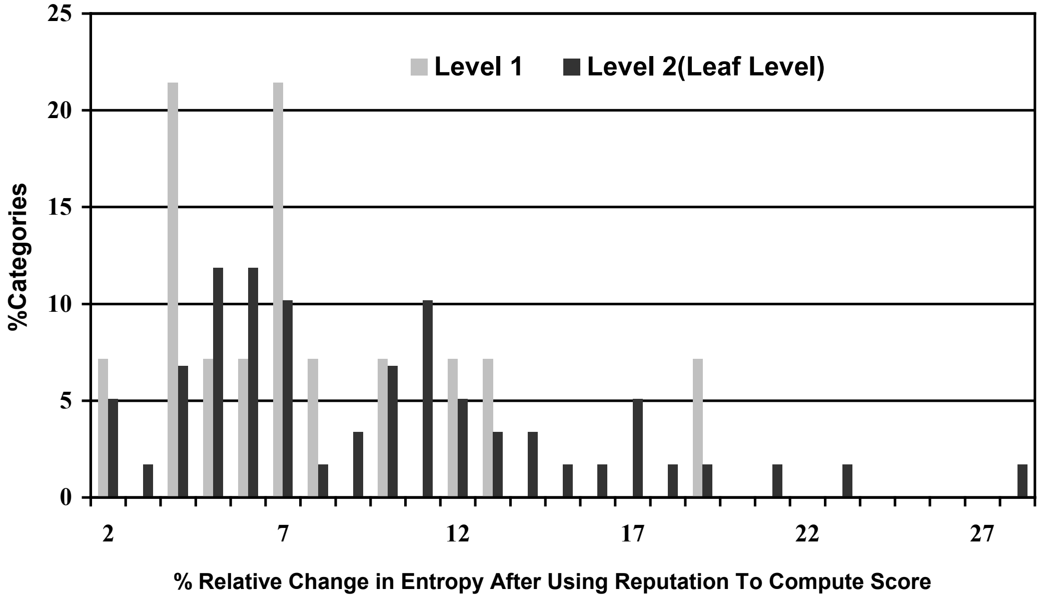


Figure 2.7 Change in score's entropy after incorporating raters' reputations in computing scores

Figure 2.7 shows that the score's entropy in ***all*** sampled categories is increased after using reputations in computing scores. The change is resulted from highlighting ratings from good raters, which supports the hypothesis that good raters are more discriminating, in general.

2.7.2.2 Comparison with Reputations on Epinions

The data used in this study were collected in summer 2000 when two special communities were qualitatively distinguished from other members on Epinions. The first community is called

"*Most Trusted Users*" who are widely trusted by other users based on a *web of trust* which propagates the trust of one user to those who trust the user. The second kind of community is *advisors* (see section 2.3.1) defined by Epinions' category managers for each category. To compare our reputation mechanism to that of Epinions, we isolate the computed *reputation ranks* of the users whom Epinions separates out into the two special communities.

(1) "*Most Trusted Users*" (*MTU*)

On Epinions, *MTU* is a global (category-independent) group. To match Epinions in our comparison, for the two hierarchical models, the *reputation ranks* in this experiment are global values at the root of reputation tree. Figure 2.8 presents the distributions of *reputation ranks* for *MTU*, computed with our four models in Table 2.2.

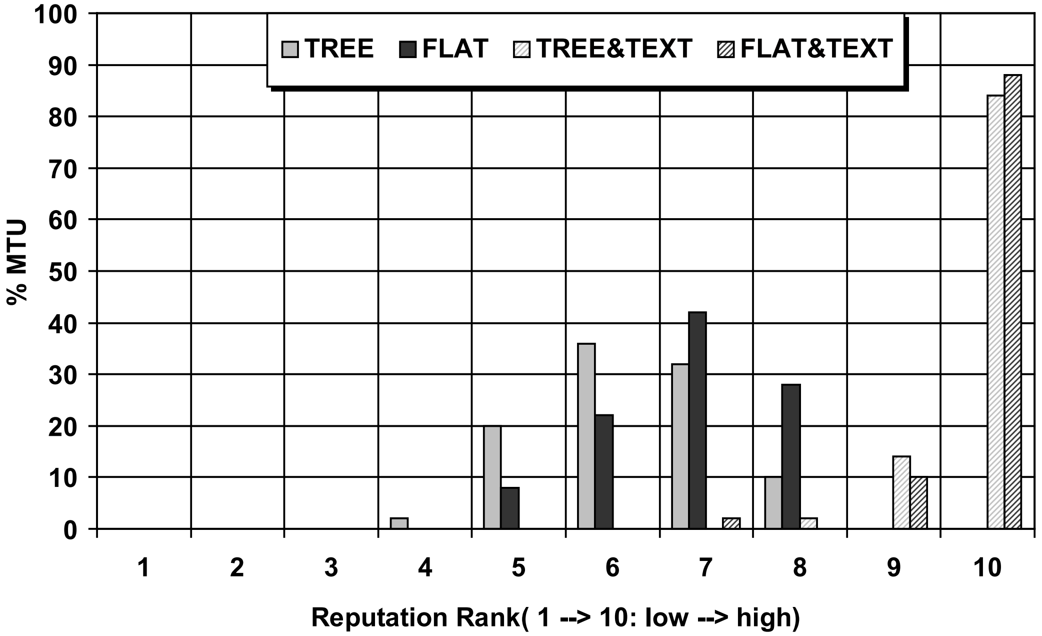


Figure 2.8 Distribution of *reputation rank* for *MTU*

From Figure 2.8 we observe that:

1.) *Reputation ranks* of *MTU* built with any of our models are located at the high ends;

2.) Interestingly, analyzing quality of text reviews makes *reputation ranks* of *MTU* much closer to the high end;

3.) The *reputation ranks* of *MTU* computed using the flat structure are closer to the high end than using the corresponding tree structure.

The second observation shows that people rely heavily on quality of text reviews to decide the trustworthiness of a reviewer. The last observation indicates that people tend to deem all data as at the same category/topic level in making overall judgments on reliability, which is natural when people need to deal with many data sources in a complicated information network. Superior to human beings, our hierarchical models can capture the information structure using the bottom-up approach to build every rater's reputation framework from the leaf categories to the root category.

The next experiment is to highlight another limitation of the *MTU* scheme due to human subjectivity and lack of ability to use the entire database of knowledge: many of the *Most Trusted Users* (*MTU*) are people who give a lot of ratings but sometimes of poor quality. In this analysis, *MTU* are ranked (compared to all users) separately by their *Category Match* and *Category Confidence* (see section 2.5.2) for the root of the reputation tree. Figure 2.9 shows the results separately for *MTU* who qualify as *Good Raters* by our reputation calculations and those who do not, using the TREE model.

One point in Figure 2.9 refers to *MTU(s)* with certain ranks of *Category Match* (*CM*) and *Confidence in CM* (*CC*). As we can see, subjectively determined *MTU* as a whole have relatively high *CC* due to their frequent rating activity on "hot" objects, however, their ranks according to *CM* are sometimes near or below the middle rank level. Our method, on the other hand, gives lower credence to those *popular raters* whose ratings' quality is not high enough at a given confidence level, which is shown by the low reputation rank (< 7) of the raters whose *Category Match* (*CC*) is low in Figure 2.9.

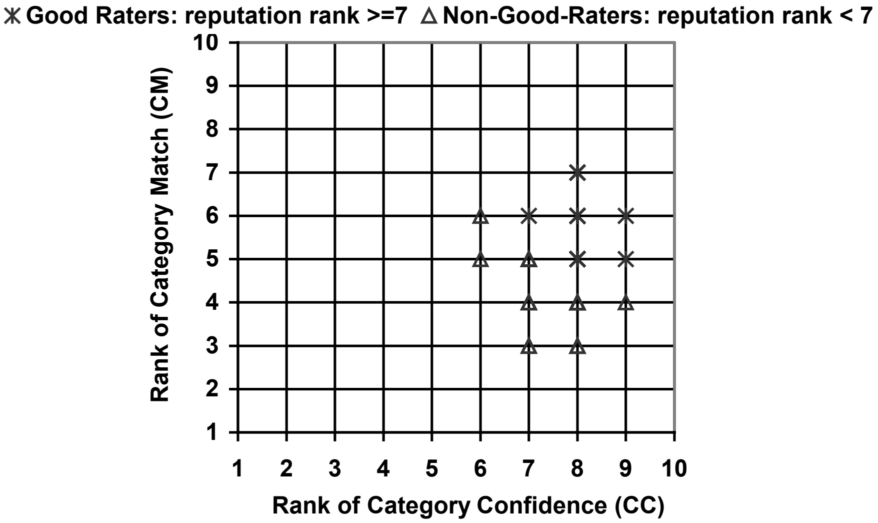


Figure 2.9 Distribution of *CM* and *CC* of *MTU*

(2) "*Category Advisors*"

Figure 2.10 shows the distribution of *reputation ranks* for *Advisors* as determined by Epinions. Using TREE model, more than 62% of *Advisors* fall into the category of *Good Raters* according to our rating system. However, this proportion achieves 96% when the quality of text reviews is included. This result is not surprising because Epinions relies on text reviews to evaluate rater's qualification.

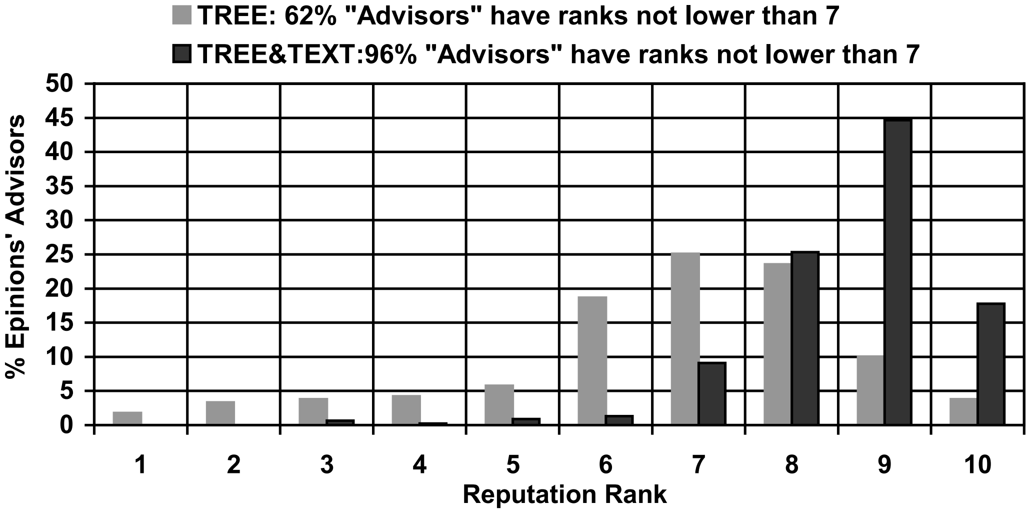


Figure 2.10 Reputation ranks of *Advisors*

2.8 Summary on Reputation System for On-line Raters

We have addressed the problem of building reputations for raters based on their rating history and that of the entire rating community. Reputations help guide users toward high quality opinions, and they also provide a standard to weigh ratings in computing overall scores for rated objects, leading to more reliable scores. Through experiments, our automated methods are shown to be more objective, precise and discriminating than existing rating aggregation systems.

The experimental results show that the reputation framework built by our statistical methods has the desirable properties with regard to temporal trend, correlation between rating experience and reputation rank, greater consistency among good raters than bad raters, and the ability of reputations (and high-reputation raters) to differentiate items more finely. In addition, special communities of raters defined manually on Epinions can also be identified using our automatic methods in most cases.

Some interesting patterns of human beings in judging trustworthiness of raters are also discovered in our experiments:

(1) People use text reviews substantially in evaluations;

(2) People tend to view different information sources as at the same level in making overall judgment;

(3) People tend to trust active raters even if their rating quality is not high.

In addition to aiding consumers in choosing products, reputations also have some social implications as follows:

(1) "Information explosion" vs. "rating scarcity"

The publication of reputations encourages content with high quality, providing a way to deal with both "information explosion" and "rating scarcity".

(2) Privacy vs. reliability

Some people do not want others to know their opinions on certain objects. This can be handled by hiding their identities and pegging them anonymously by reputation instead. Without a reputation scheme, anonymity often degrades the reliability of people's opinions, and reputation information helps protect privacy without loss in reliability.

(3) Robustness of rating aggregation methods

Boosting scores by fake ratings is a typical cheating behavior in rating-based reputation schemes. With our reputation framework, intruders need to gain sufficient reputation to make their attack work. They will need to "attack" other rating profiles of items besides the product they are actually interested in, and understand how to gain high reputations based on our methods. Giving ratings (e.g. fake ratings) that deviate from those of the rating community leads to lower reputation, which means that such attacks must be conducted by not just a few but many conspirators (end users). Also, given that the rating sites are open to all users, as new ratings come in, the rating profiles change, so maintaining a reputation artificially is a dynamic and expensive affair.

Chapter 3

Content Delivery in Publish/Subscribe Applications

Starting from this chapter, we present the approaches to enhance content delivery service using user-provided preference profiles. This chapter first introduces publish/subscribe model, the basic communication model used in the content delivery service studied in this thesis. After that the related work on publish/subscribe system is reviewed. Focusing on publish/subscribe applications that involve intensive content delivery, we introduce the importance of content placement problem in publish/subscribe systems and identify the major problems that this thesis addresses. The last section of this chapter discusses the related work on content delivery and web caching techniques.

3.1 Communication Paradigm of Publish/subscribe Systems

Publish/subscribe (pub-sub) is a communication paradigm for information producers (publishers) and information consumers (subscribers). Different from the conventional client-server communication model for one-to-one interaction, the publish/subscribe model supports many-to-many interaction between publishers and subscribers. Specifically, the publish/subscribe model is a loosely-coupled model in three aspects: space decoupling, time decoupling and flow decoupling [Eugster ’03]. Firstly, space decoupling indicates that publishers and subscribers do not need to know each other and hence the two parties do not need to indicate the address of the destination in their communication. Secondly, pub-sub model is time-decoupled, which means that the two parties do not need to connect to the network at the same time; in other words, publishing and consuming can be done asynchronously. Finally, flow decoupling means that polling for communication is not in the main flow of the participating entities, and hence the pub-sub communication does not block the main flow of control at either the publisher side or the subscriber side.

Pub-sub communication can be performed within a local network, but in many important pub-sub services, the publishers and subscribers are globally distributed. Dedicated publish/subscribe middleware should be designed to support Internet-scale scalable publish/subscribe services. Both publishers and subscribers talk to each other via the dedicated pub-sub system: information consumers declare their information needs to a publish/subscribe system, publishers generate events to the pub-sub system that notifies the subscribers if the published content matches the users' subscriptions.

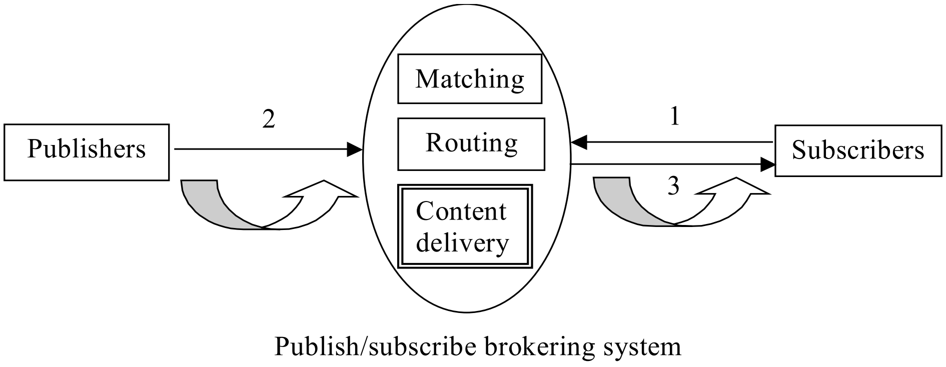


Figure 3.1 Architecture of a publish/subscribe system

Figure 3.1 outlines a conceptual architecture of a publish/subscribe system. Publishers and subscribers (end-users) are connected via the publish/subscribe system. When a publisher generates the content of some subscribers' interest, the publish/subscribe system notifies the subscribers. Notification services are usually implemented through three basic communication steps as labeled in the figure:

(1) Users subscribe, announcing their interests to the system;

(2) Producers publish contents to the system;

(3) The system notifies the users whose subscriptions match the contents.

The communication between publishers and subscribers is driven by matching content and users' interests described as subscriptions. A typical system consists of a matching engine and a routing engine. After content is published into the system (flow 2), the matching engine finds the users who are interested in the events according to their subscriptions, and the routing engine delivers events to those users. The matching engines and routing engines may be centralized or distributed, and are often coupled.

3.2 Related Work on Publish/subscribe Systems

As an asynchronous and content-driven communication paradigm, publish/subscribe is used as a platform to build many systems and applications. The research in publish/subscribe systems can be classified as network layer communication protocols, infrastructure design, and applications. Most research about publish/subscribe systems is focused on the issues surrounding the system infrastructure design. Specifically, many systems examine event routing and efficient matching [Makwana '02, Carzaniga '98, Altinel '00, Fabret '00, Fabret '01, Eugster '00].

According to the expressiveness of subscription language, pub-sub models can be classified as topic-based, content-based and type-based. Topic-based publish-subscribe systems allow users to express their interest at the topic or subject level [TIBCO '99]. Type-based pub-sub systems require subscriptions to be defined using object-oriented language and enable the integration of middleware and language in an object-oriented programming environment [Eugster '00].

Content-based publish-subscribe systems are more powerful and flexible in the sense that users can define their interest in content using predicates on keywords. Siena [Carzaniga ‘98] is a content-based distributed publish/subscribe system. Carzaniga et. al. define and use a coverage relation for messages and subscriptions to achieve high scalability. Cao and Singh propose a content-based routing scheme that combines dynamic content-based filtering in routers and multicasting in static and pre-defined multicast groups [Cao ’04]. Semandex [Makwana ‘02] is a content-routing network to deliver messages from the producers to the subscribers by matching messages against the aggregated subscription profiles in each router.

Pub-sub systems can be centralized or distributed. While centralized systems have advantage in maintainability, distributed systems provide better scalability and availability. Gryphon is a full-fledged distributed publish/subscribe system developed by IBM researchers [Gryphon ’98]. Gryphon supports both topic-based and content-based publish/subscribe services. In addition to the basic modules such as multicast-based routing algorithms [GryphonR ’99, GryphonR ’00] and matching algorithms [GryphonM ’99], Gryphon project also proposes the algorithms and the architecture to handle durable subscriptions and congestion control in scalable publish/subscribe services [GryphonD ’03, GryphonC ’03].

Other interesting issues around publish/subscribe systems include security and economic issues. Wang et. al. investigate the security concerns within a pub-sub service that must handle information dissemination across distinct authoritative domains [Wang ‘02]. Evans et. al. propose a mechanism to collect payments from subscribers and distribute them to publishers [Evans ‘03].

The publish/subscribe paradigm supports various academic and commercial applications. Elvin is an event notification system that helps people form dynamic community [Fitzpatrick ‘00]. Bornhvd et al. [Bornhvd ‘00] propose an Internet-scale shopping platform by using a pub-sub system to aggregate the auction information across multiple sites. Pub-sub can also be used as a communication paradigm in multi-agent system [Skarmeas ‘98] and in sensor networks [Huang ‘01]. Peer-to-peer applications form another area to apply publish/subscribe communication model. Scribe [Scribe ‘01] is a topic-based publish/subscribe system built on top of Pastry [Pastry ‘01], a generic peer-to-peer object locating and routing substrate overlay on the Internet. Heimbigner [Heimbigner ‘00] presents publish/subscirbe middleware to achieve the functionalities such as searching of a Gnutella-like peer-to-peer system.

3.3 Content Placement Problem in Content-intensive Publish/subscribe Services

In the literature, most work on publish/subscribe systems examines event routing and efficient matching. However, the storage management problem in content-intensive publish/subscribe services, or using publish/subscribe information to improve content distribution and caching, has not been adequately studied to the best of our knowledge. The content delivery module is usually ignored because the existing publish/subscribe applications assume subscribers are only interested in short messages rather than large-size contents. However, this assumption does not hold for many applications that have intensive information exchanged.

An example is the notification services at news sites. A user indicates the categories or keywords of the news of interest; the news site notifies the user with a list of titles when it publishes news that matches the user’s subscription. If the user wants to read an article in the list, the user requests the actual content from the origin site. In this type of news notification services, the object of interest (to which the notification might only carry a link) may embed long texts, images and video/audio streams.

3.3.1 Content Delivery for Content-intensive Pub-sub Services

In content-intensive publish/subscribe services, an important performance issue is user perceived response time to receive the contents of their interest. There are two simple content delivery approaches for publish/subscribe services. One approach stores all the contents at the publisher side until users request them at access time. Another method delivers and stores all the matched contents at the subscriber side once the content is generated. To guarantee short response time, the former approach assumes low network latency, while the latter assumes unlimited storage capacity at the user site.

This study considers globally distributed publish/subscribe services in which intensive content is generated continually. Besides that, we do not make specific assumptions about the system capability and network capability. For the existing Internet, latency constraint cannot be ignored in general, which implies the inefficiency of the approach that delivers contents only on demand. Two reasons make the latter approach that stores all contents at user side until users read them unrealistic. To serve users' requests most efficiently, the contents requested by users should be stored in the main memory which is definitely constrained in the memory size. Alternatively, content can be stored in and served from the low-cost and large-size disks of the proxy servers that are close to end-users. Even if content is stored in the disks, the storage capacity may still become a problem because the time-decoupling characteristic of the publish/subscribe model implies a lot of contents may be accumulated at user-side before the users request them, especially in the content-intensive applications. Therefore, intelligent and dynamic content delivery strategies must be developed.

To enhance content-intensive publish/subscribe services, this thesis addresses content delivery and caching module in distributed publish/subscribe middleware. To the best of our knowledge, this is the first effort on enhancing content delivery in the context of publish/subscribe services. After step 3 in the notification service outlined in Figure 3.1, the notified users may choose to read the actual content of the event; the content delivery module in Figure 3.1 is in charge of delivering content to subscribers who request them.

We take the “pushing to edge” philosophy in designing our content delivery module for publish/subscribe middleware system. Our content delivery module is a distributed overlay network of content servers that are deployed close to the end-users. Each content server connects to a group of users/subscribers. A content server aggregates its users’ subscriptions and processes notifications for its users. Each content server also serves as the cache that is consulted when one of its users accesses content. In the following discussion, the terms proxy, proxy server and server are used synonymously for content distribution server.

3.3.2 Research Issues in Content Delivery for Pub-sub Applications

The information needs of content consumers form a key to driving content delivery over the Internet. Typically, these information needs are determined based on access patterns and pre-determinations of popular resources. Web-based notification services are based on users’ subscriptions, which are statements of interest. The stated interest can therefore also be used as a basis for content delivery and caching. Little exploration of this use has been done.

The loose-coupling characteristics of publish/subscribe services provide two opportunities for improving content delivery and caching in publish/subscribe scenario. Subscriptions provide a priori knowledge about users’ interest in contents. Once content is published, the matching engine can decide which users’ pre-defined preferences match a given published event. The matching result is useful for estimating the usage pattern of the content at each server, which creates an opportunity for proactive content delivery at publish-time. Delivering content before users request the content can be successful because information producing and consuming occur asynchronously in publish/subscribe services. In addition, after users start to request the contents at a server, the access pattern and the subscription information of the users can be analyzed together to decide the value of each page on the fly.

Some interesting issues about content delivery arise in publish/subscribe services:

1.) How to use pub-sub information in designing content delivery and caching algorithms;

2.) How to extend the methods to benefit users of non-publish/subscribe services;

3.) What are major performance metrics for comparing different approaches;

4.) How to develop realistic workloads for evaluation, a major challenge given that publish/subscribe workloads are not generally available.

3.4 Highlights of Our Study on Subscription-enhanced Content Delivery

We investigate the content delivery and caching problem under two scenarios: for pure publish/subscribe service and for an integrated service shared by publish/subscribe services and non-publish/subscribe services. The former assumes that the content delivery module serves only the publish/subscribe users who make accesses to content based on notifications received from the publish/subscribe system (i.e. a user does not access any page unless a notification for it has been received). The latter case assumes that our content delivery module serves both publish/subscribe users and those who access web content not via publish/subscribe services, in which case publish/subscribe information is used as one more source of information for facilitating content delivery for all users. In other words, in the second scenario, a user of each content server may access a given piece of content either based on notifications or based on general browsing as is done without a publish/subscribe environment.

This thesis presents a set of content delivery strategies that use publish/subscribe information dynamically. The different approaches can be classified along two axes, which also expose the key design issues: (i) *when* is the best opportunity for placing a page into a cache server; (ii) *how* (on what basis) should the placement and replacement decisions be made at evaluation time of pages at a server. The two major possibilities for *when* are (a) at publish/match time, i.e. when a page is determined to match certain subscriptions and (b) at access time, as in traditional caching systems. The two major possibilities for *how* are (a) based on access patterns only, as in traditional caching systems, and (b) based on matching results between subscriptions and content.

A major challenge in such a study is developing workloads, given that no real-world publish/subscribe workloads are available for such studies. We therefore have developed workloads based on studies of observed access patterns at busy sites, extrapolating from there to publish/subscribe workloads. In particular, to study the performance of our approaches, we simulate news delivery to subscribers who are geographically distributed. The publishing pattern and the access dynamics are simulated according to a study of accesses to one of the busiest media sites, MSNBC [MSNBC].

In our experiments, the major performance metric is the hit ratio in the local proxy servers, since the major goal of this work is to reduce the response time perceived by end-users. Using a purely access-based caching approach as the baseline, our approaches improve the hit ratio dramatically (and thus reducing the response time perceived by users). This is even true when the content delivery service serves both the publish/subscribe users and other users in which case predefined subscription information does not reflect the actual users' access patterns perfectly.

Our investigation also measures the total traffic as the bandwidth consumption for transferring contents from the publisher site to the proxies of subscribers. Although improving hit ratio should tend to reduce traffic, this metric is useful because the reduction in traffic resulted by higher hit ratio is offset by the traffic of proactive content placement. Existing prefetching techniques usually have to trade bandwidth for hit ratio. In the pure publish/subscribe scenario, our best approaches yield higher hit ratio than the baseline algorithm while keeping the traffic overhead comparable.

3.5 State of the Art of Content Delivery Network and Web Caching

Content delivery network (CDN) aims at facilitating distribution of web content by using globally distributed servers. From the perspective of end-users, CDN provides faster and more reliable access to web contents; for the content providers, CDN helps shift the workload and remove “flash crowd” in requests at the providers’ origin sites; the Internet service is benefited by avoiding unnecessary traffic between hosts. Our content delivery scheme carries the same goal as a CDN under the scenario of content-intensive publish/subscribe services.

Content delivery networks have evolved in three major stages. The earliest systems were based on smart DNS (domain name service) and passive catching. Then, the concepts of CDN and cooperative caching system were proposed. Nowadays systems put emphasis on dynamic content processing. An important trend is “pushing to edge” that puts the processing power of CDN to the servers at the Internet edge. For example, some CDNs like Akamai [Akamai] adaptively assemble content at edge servers which are deployed close to the users.

A content network consists of many components, including a caching module, a proactive content placement module, a content-based routing system, the components to achieve global and/or local load balancing, etc. An important component in CDN is load balancing. Most commercial products and research in CDN focus on request redirection to achieve load balance among servers and thus reduce the response time [Akamai, Digital Island]. Hashing is the basis of many request redirection schemes [Karger ‘97, Wang ‘02].

Instead of designing a complete general-purpose content delivery network, this thesis focuses on content placement and caching modules for special-purpose content delivery in publish/subscribe services. Our approach also uses the "pushing to edge" philosophy by serving contents from local content servers close to end-users.

3.5.1 Web Caching and Caching in CDN

Web caching has been widely applied to distribute content via the Internet. Web caching passively keeps the most useful information in a capacity-limited proxy server.

Many caching replacement algorithms have been presented in the literature, from classical Least-Recently-Used (LRU) algorithms, Least-Frequently-Used (LFU) algorithms, to more complicated approaches that consider other factors such as fetch cost. A representative cost-aware approach is the GreedyDual-Size algorithm that combines factors such as temporal locality and popularity as well as fetching cost and page size in caching replacement [Cao '97]. GreedyDual* is a generalization of GDS and balances the effects of long-term popularity and short-term reference correlation in a reference stream [Jin '01].

Caching is also an important module in modern CDN. The caching systems in CDN are usually cooperative systems. Gadde et al. [Gadde '00] study how to derive the hit ratios for interior caches in a multi-level cache hierarchy of CDN servers in which misses in a leaf cache are redirected to a parent cache. An important conclusion in the study of Gadde et al. is that there may exist a natural limit to the marginal benefits of redirection-based CDNs, since the hit ratio in proxy caches increases dramatically as ISPs serve larger user communities. The results of a recent experimental study show the superiority of globally deployed proxy caches compared to a separate CDN [Saroiu '02]. To the best of our knowledge, the caching algorithms in the existing CDNs rely on request pattern analysis as conventional caching schemes.

Two characteristics of conventional caching approaches form the fundamental differences between catching and our content delivery algorithms. First, caching mechanisms passively store only the contents that have been requested by users. Our approaches place contents proactively before users request them. Secondly, the replacement algorithm in caching approaches considers factors such as access patterns, while our approaches incorporate pre-defined user interest as well as access patterns in evaluation of pages.

3.5.2 Prefetching: Client-initiated Proactive Content Placement

Prefetching is used to proactively pull information from an original site to a proxy server [Deolasee ‘01, Duchamp ‘99, Venkataramani ’01 TR], or from a proxy to a browser cache [Fan ‘99]. A successful prefetching algorithm loads contents into a local cache before users request them. The success of prefetching relies on the prediction accuracy about usage patterns. The major information used by prefetching algorithms is the dependency between the pages that have been requested and the pages that can be fetched from a source. The dependency can be inferred from access pattern, page content, link structure, etc.

Many prefetching techniques predict which pages will be requested based on an access sequence in history, using Markov models as analysis tools. Fan et. al. [Fan ‘99] propose a prefetching mechanism by mining the reference dependency between pages from users’ access sequences at a proxy. However, access history is not the only knowledge used in prefetching techniques. Some personal browsing agents prefetch the pages that link to the page a user is reading or based on the similarity in pages’ contents [Chi ‘01, Lieberman ‘95]. Combining access information and link structure together, Duchamp proposes a mechanism to pull the embedded components that are popularly referenced when a given page is read [Duchamp ‘99]. Venkataramani et al. present a threshold-based prefetching algorithm that accounts for the long-term popularity of pages and the content’s update rate [Venkataramani ’01 TR].

The content delivery approaches proposed in this thesis utilize pre-defined user interest as a new information source for predicting the usefulness of each page for a server's cache.

3.5.3 Replication: Server-initiated Proactive Content Placement

Rabinovich [Rabinovich '98] differentiates replication from caching by whether the placed content would be queried by users. Caching stores the content that has been requested by some users, while replication schemes place content based on predictive usage patterns and hence the placed content may not be requested by any user eventually. We believe another dimension to distinguish replication and caching is whether the content placement is performed on behalf of publishers or users. The replication scheme for the publishers can make use of publisher-side information or global information about the access patterns and traffic in a content delivery network, while user-side caching approaches have only local information.

3.5.3.1 Server-initiated Pushing

Server-initiated multicast is a kind of dynamic replica placement according to the categorization of Rabinovich [Rabinovich '98]. Besravros [Besravros '95] proposes a pushing algorithm that places the most popular pages at the layer closer to the end users in a hierarchical caching system. In the system presented by Gwertzman and Seltzer [Gwertzman '97], the popular pages are automatically pushed to the proxy servers that request the pages frequently. Similar to some link-based prefetching techniques, a server can actively push embedded objects when a page is requested [Tang '00].

Server-based multicast is also used in server-based invalidation for frequently updated objects [Deolasee '01, Li, D. '99]. Deolasee et al. combine push and pull adaptively according to users' requirements for the consistency of dynamic contents and other factors such as the capabilities of servers and proxies [Deolasee '01].

3.5.3.2 Replication Schemes in CDN

As a dedicated engine, CDN is in charge of content replication on behalf of publishers and the replication is done among the multiple servers of CDN. The replication problem in CDN is usually referred to as content placement problem.

There is an increasing interest in the placement problem in CDN. The previous work done in CDN has mainly concentrated on optimum solution for restricted scenarios. Most of the study of the optimum placement in CDN is space-constrained. Kangasharju et al. [Kangasharju ‘01] formulate an optimum content replication problem as the problem of minimizing the average number of network hops to fetch a copy, assuming the locating algorithm always gets a copy from the nearest server holding a copy. This is a typical formation of the placement problem in CDN. The optimum placement in an overlay network with a graph topology has been proved to be NP-hard, while there exist polynomial solutions for other topologies like trees [Krishnan ‘00, Li, B. ‘99]. Cidon et al. [Cidon ‘01] present an optimum placement solution in a distribution tree, but the work assumes a static environment in which all the requests are given precisely in advance and the network does not change. The assumptions such as having global and static information used by optimum solutions make the solutions infeasible for web-applications, which are usually featured in highly dynamic content publishing and requests.

Researchers design and evaluate placement heuristics for realistic applications. Kangasharju et al. [Kangasharju ‘01] propose four heuristics, but the best one needs to exploit global knowledge about the network topology, the global reference distribution and the global content image at different times. Qiu et al. [Qiu ‘01] propose several heuristics to choose *M* replica sites from *N* candidates for a given site, also assuming a relatively stable reference pattern at the candidate sites.

Space is not the only constraint in content placement problems. Venkataramani et al. [Venkataramani ‘01] study the bandwidth-constraint placement problem. They present a solution whose expected response time is within a constant factor of the optimal placement if the information objects have uniform size. However, the algorithm is not designed for a highly dynamic environment in which the object update rate is high and demand-readings proceed with publishing in parallel.

Although it is usually assumed that proactive replication mechanisms are superior to caching approaches, Karlsson and Mahalingam [Karlsson ‘02] argue that caching works at least as well as replica placement algorithms, if the cache storage is ample and the replacement algorithm runs periodically rather than after each access.

3.5.4 Combining Passive Caching and Proactive Content Placement

Chang and Chen [Chang ‘02] combine caching factors and prefetching factors into a single replacement algorithm, and prefetching is also based on the dependency among page references.

For combining replication/placement and caching, Korupolu and Dahlin propose a “static partition” approach that divides a server’s cache into two portions and runs placement and caching algorithms on the two portions separately [Korupolu ‘02]. The approach is analogous to the Dual-Caches Partitioned by Operation algorithms (DC-O) that we propose in chapter 4. However, several major distinctions exist between the two. Our DC-O takes into account stated user interest rather than being purely based on request distribution. For the placement portion, our push-time placement is triggered by content generation and matching the content to users’ subscriptions, while placement is performed periodically in Korupolu and Dahlin’s system. Moreover, DC-O re-partitions a server’s cache according to the access and publishing pattern.

3.5.5 Interesting Problems in Content Delivery and Caching Algorithms

Two problems have not been investigated adequately in the area of content delivery and caching algorithms. One has to do with the information or knowledge that can be used in the algorithm design. In existing content delivery systems, caching, prefetching and pushing are mainly based on inferred user interest. With publish/subscribe applications becoming more and more popular, publish/subscribe information such as user preference stated via subscriptions is a new source for intelligent content delivery. Another important problem in the study of content delivery is a system's scalability and adaptability in a highly dynamic environment.

This thesis addresses how to exploit the stated user interest in conjunction with the inferred user interest from their access patterns to facilitate content delivery. Our approaches use a distributed network of content servers each of which is autonomous in managing its storage using local subscription and access information only. In this study, the proxy servers act as both demand-side caches and the supply-side caches [Gadde '00], and there is no cooperation between the proxies. This work focuses on the coordination between a publisher and a content server. Therefore, our approaches are scalable in a globally distributed system that encounters dynamic content publishing and request patterns, and network conditions. Exploring the algorithms for cooperative content servers is one topic of future work to extend this study.

Chapter 4

Content Delivery Algorithms Using Subscription

This chapter proposes our content delivery algorithms for caching contents at servers that are close to end users. We first outline the architecture of our content delivery engine for publish-subscribe services, and then describe the specific content delivery algorithms.

4.1 Architecture of Our Content Delivery Engine

For serving users' requests from the Internet edge, we assume the content servers as the endpoints of the publish/subscribe systems as shown in Figure 4.1. Every content server collects the subscriptions from its local users and uploads the subscription information to the publish/subscribe system. From the perspective of the publish/subscribe service, the content servers become the pseudo-users each of whom represents the subscribers in a certain area.

In this study, we assume a published event consists of two separate parts: a *header* which is the summary (e.g. news title) or the meta-information on the event, and *content* which is the body of the event. The header of every event, usually called *event* in publish/subscribe service, should always be stored in the content server that is notified as assumed by general publish/subscribe service. Our content delivery algorithm deals with the delivery and caching of large-size event body. When an event is published to the publish/subscribe system, the matching engine decides a list of subscribers whom should be notified with the generation of the event, and the routing engine delivers the event to the content servers of those subscribers. Each notified content server runs the content placement algorithm to decide whether only the header or the whole event including the content should be stored in the server's cache.

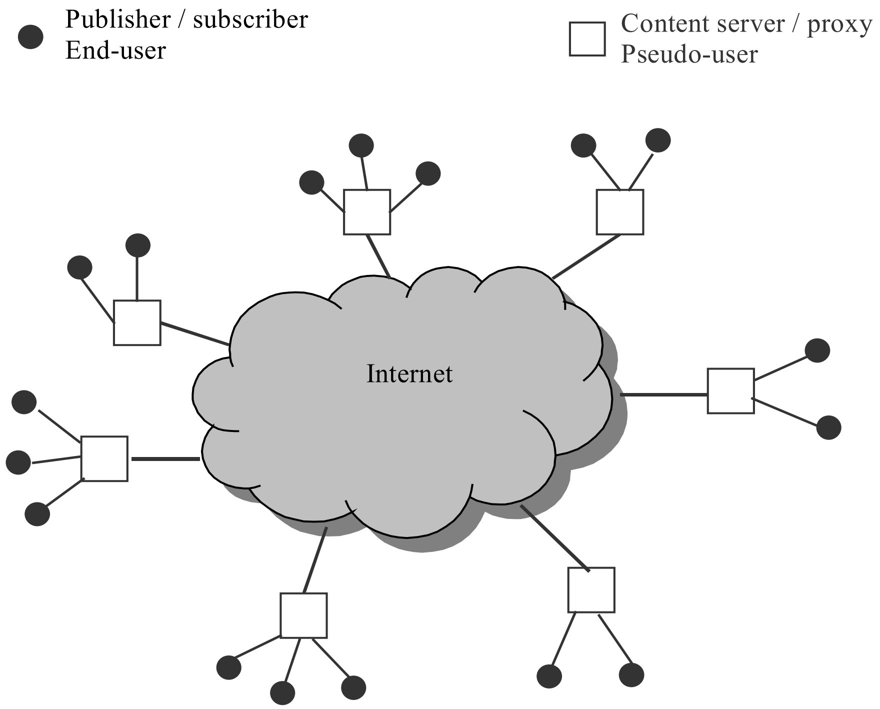


Figure 4.1 Deployment of content delivery engine for publish/subscribe services

When a user requests the content associated with a certain event, the local server first checks the local cache. If the content is in the local cache, the server replies the user request with the content; otherwise, the server fetches the content from the publisher of the content.

4.2 Categorization of Algorithms

The information needs of content consumers form the key to intelligent caching and content delivery over the Internet. Subscriptions provide the basis for estimating future requests from subscribers. This estimation can be done at publishing time by matching the content to subscriptions from all users, which creates an opportunity for early content distribution (the matching of content to subscriptions may be done in an intermediary publish-subscribe service that sits between the user and the content source, and may be based on the entire content or subsets of the content or metadata).

On the other hand, subscription information by itself is insufficient for designing efficient content delivery algorithms. This is because subscriptions just imply the static distribution of subscribers' requests in terms of numbers but not the dynamic access patterns of users. Moreover, pre-defined subscriptions may not even reflect the actual request distribution of users perfectly. For example, subscribers may not read every page that matches their subscriptions. Therefore, the access patterns of users provide complementary knowledge to subscriptions for estimating the actual temporal and frequency request patterns on the fly.

The focus of this study is on using subscription and access information together in both proactive content placement at publishing time and on-demand caching.

As discussed earlier, content delivery strategies can be classified along two axes, namely *when* and *how* pages are evaluated for placement in caches. Regarding *when*, the content distribution engine has two obvious opportunities to deliver content from a publisher to a proxy server. In the first case, the publisher proactively forwards a page to the proxy for potential placement in the cache when the matching engine determines that the content of the page matches some of the subscriptions of some users that contact that proxy. This approach, which we call the *publish-time* strategy, assumes that the subscribers are likely to request the page later after receiving the notifications. The second scenario, which we call the *access-time* strategy, is like traditional caching and is based on the fact of rather than the prediction or likelihood of users' accesses to a page. A content distribution engine can apply one or both of publish-time and access-time strategies.

Since a server is limited in physical storage capacity, when a page is delivered to a server in either the publish-time or access-time approaches, the server may need to replace some content at the local server if the server's cache is already full. Replacement is based on values assigned to pages. Regarding *how* the value is determined, it can be done based on (predictive) subscription and matching information or on actual access information.

This thesis focuses on the following interesting combinations of when and how evaluations are made for page placement in caches:

1.) Access-time strategy based on access pattern only

2.) Publish-time strategy based on subscription and matching information only

3.) Publish-time and access-time strategy based on subscription and access information

The first combination is the traditional caching approach and is the baseline used in this study. The second combination is a simple way to apply the subscription information in content delivery for publish/subscribe applications. The emphasis of this thesis is the third class of schemes that combines both proactive publish-time and on-demand access-time placement using pre-defined subscriptions as well as analysis on dynamic access patterns.

4.3 Algorithms for Pure Publish/subscribe Applications

In this thesis, a pure publish/subscribe system is defined as a system in which the only communication between content publishers and content subscribers is via a notification service. Therefore, a subscriber requests content only if the content matches the user's subscriptions and thus the user receives a notification message about the content. In the following discussion, the content associated with an event is called a "*page*".

4.3.1 Access-time Strategy Based on Access Patterns Only

As mentioned in the categorization of the content delivery algorithms, conventional caching approaches belong to this category. There is a lot of research in caching algorithms. This thesis uses a replacement algorithm called Greedy-Dual* (GD*) [Jin '01] as the baseline algorithm.

GD* generalizes Greedy-Dual-Size (GDS) [Cao '97] by adjusting the relative worth of long-term popularity and short-term temporal correlation of accesses. In an experimental study, GD* demonstrates superiority in terms of hit ratio as compared to LRU, GDS and LFU-DA [Jin '01]. In GD*, the value of a page *V*(*p*) is determined by the access frequency, the access recency, the cost to fetch a page, and the size of the page, as represented in Equation 4.1.

$$V(p) = L + \left(\frac{f(p) \cdot c(p)}{s(p)} \right)^{1/\beta} \qquad \text{- Equation 4.1}$$

Where

L: inflation value of a cache to capture the access recency for each page

$f(p)$: total number of accesses by the users of a cache on the page

$c(p)$: cost to fetch a page from the publisher of the page

$s(p)$: page size in bytes

β: balance factor of long - term popularity in terms of $f(p)$ and temporal correlation of references in terms of L

In the implementation, the reference count of a page, *f*(*p*), is discarded when the page is evicted, as in the In-Cache Least-Frequently-Used (LFU) [Jin '01]. Based on the experimental study about *GDS* replacement algorithms [Cao '97], the network hops from the origin site is a good metric for the cost to fetch a document. In our implementation, the network distance to the origin publisher is used to measure the cost to fetch a page for a given proxy, where the network topology of proxy servers and the publishers is a random graph built using the Internet topology simulator BRITE designed by Boston University [BRITE]. The constant parameter β is set manually according to preliminary experiments on the workloads (see chapter 5).

The pseudocode of the replacement algorithms in GD* is shown in Figure 4.2. On an access hit of page *p*, the replacement algorithm GD* increases the reference count *f*(*p*) and recalculates the page's value based on the current inflation value *L* for each page. To make the algorithm more efficient, in our implementation the recalculation of a page's value is only carried out when a replacement in a cache is triggered. On a cache miss, all the pages residing in the cache are sorted by values, and pages are evicted from the cache until there is room for the requested page; the inflation value *L* is set to be the value of the page that is evicted last.

```
$L \leftarrow 0.0$
For each request in turn:
The current request is for page $p$:
  If $p$ is already in memory
      $V(p) \leftarrow L + \left( \frac{f(p) \bullet c(p)}{s(p)} \right)^{1/\beta}$
  Else
    While there is not enough room
      $L \leftarrow \min \{ V(k), k \in \text{pages in the cache} \}$
      Evict $q$ s.t. $V(q) = L$
    Bring $p$ into cache and $V(p) \leftarrow L + \left( \frac{f(p) \bullet c(p)}{s(p)} \right)^{1/\beta}$
End
```

Figure 4.2 Replacement algorithm in *GD**

4.3.2 Publish-time Strategy Based on Subscriptions

At the publishing time of an event, the matching results between the event and users' subscriptions provide a good estimation on the request patterns from the content servers in the future. Specifically, the number of subscriptions matching the event on a server implies the total number of accesses of the event at the server. This static subscription information can be used to evaluate pages in a publish-time replacement.

In addition to the subscription information, the publish-time evaluation also considers other meta-information about a page, for example, the cost to fetch a page from a publishing site to a server and the page's size. Equation 4.2 is used to evaluate a page in a publish-time replacement. Since only the static information regarding a page's value is available at the publish time, the evaluation does not incorporate any temporal information.

$$V(p) = \frac{f_S(p) \cdot c(p)}{s(p)} \qquad \text{- Equation 4.2}$$

Where
$f_S(p)$: the number of subscriptions matching the content of page p
$c(p)$ and $s(p)$ have the same meaning as in the equation 1

When an event is generated and routed to a destination proxy (based on matching the event and users' subscriptions), the new page associated with the event is evaluated and compared to the existing pages in the proxy. If it is necessary to evict some pages from the local cache to make room for the new page, the pages whose values are lower than that of the new page are candidates. The candidate pages are sorted, and the least valuable page is evicted in each run until the available storage on the proxy is large enough for the new page.

The pseudo-code in Figure 4.3 shows the replacement algorithm in the subscription-based publish-time strategy, denoted as SUB in this thesis.

```
For each page published to the server in turn:
The current page to store is page $p$:
  $V(p) = \frac{f_S(p) \cdot c(p)}{s(p)}$
  If the available storage in the cache $\le Size(p)$
    $V_{\min} \leftarrow \min\ \{ V(k), k \in \text{pages in the cache} \}$
    While ($V_{\min} \le V(p)$) AND (no enough room)
      Evict $q$ s.t. $V(q) = V_{\min}$
      $V_{\min} \leftarrow \min\ \{ V(k), k \in \text{pages in the cache} \}$
  If the available storage in the cache $\ge Size(p)$
    Bring $p$ into cache with value as $V(p)$
End
```

Figure 4.3 Replacement algorithm at publish-time in *SUB*

Different from the access-time replacement algorithm, the publish-time placement possibly decides not to store a new page if the total size of all the candidate pages (i.e. the pages whose values are smaller than that of the new page) is smaller than that of the new page. Since SUB is a publish-time-only placement strategy, on a cache miss, SUB fetches the requested page from the publisher and forwards the page to the user *without caching the page in the local server*.

4.3.3 Publish-time and Access-time Schemes Using Subscription and Access Pattern

Within the class that performs both publish-time and access-time placement, we developed different approaches to analyze the subscription and access information. The approaches in this category have two independent placement modules that are executed at publish-time and access-time respectively. However, the two placement modules use the same replacement algorithm.

Similar to SUB, the replacement algorithms decide whether to store a page on a server purely based on the value of the page. The publish-time placement fails to store a new page if its size is larger than the total size of all the candidate pages for eviction. On a cache miss, if the value of the requested page is not high enough to stay in the cache, the page is discarded immediately after being forwarded to the end-users.

Within this framework, the evaluation function used in the replacement algorithm can combine the subscription and access information together in several ways. Analog to the basic categorization criterion for caching approaches, our algorithms can be classified as frequency-based and frequency-recency-based approaches.

4.3.3.1 Frequency-based Approach

To explore the usefulness of the number of subscriptions only, we develop a new evaluation function that relies on the prediction on access frequency as well as other static factors regarding a page's value. This evaluation function is defined in Equation 4.3. The approach using this evaluation method is referred to as *Subscription-Request* or *SR* in this thesis. *SR* can be viewed as an analog to LFU algorithms and as an adaptive version of SUB.

$$V(p) = \frac{f(p) \cdot c(p)}{s(p)} \qquad \text{- Equation 4.3}$$

Where

$f = s - a$

4.3.3.2 *GD*-based Approaches*

Since GD* provides a general framework to combine several factors related to a page's value, it is used as the basis of the evaluation function to incorporate the subscription information. Two evaluation functions are developed based on GD*.

For a given page at any given time, the number of end-user subscriptions that match the page is an indication of the total number of accesses to the page in the whole application, while the number of accesses in the past exhibits users' actual interests in the page. A direct way to combine the subscription and access information is to use the sum of the two. The intuition is that a page is valuable if many users' stated interest matches the page and the page is popularly used by users. The evaluation function based on this idea is Equation 4.1 after replacing the frequency factor *f*(*p*) by the sum of the number of subscriptions and of the accesses as in Equation 4.4:

$$f(p) = s + a \qquad \text{- Equation 4.4}$$

Where

s : the number of subscriptions matching page p

a : the number of accesses of page p

The above approach, denoted as *Subscription-GD*-1* or *SG1* in this thesis, ignores the relationship between static pre-defined user interest and dynamic user access. Ideally, if every subscriber reads any page that matches his/her subscription exactly once, the difference between the number of subscriptions and the number of past requests at any time is exactly equal to the number of future references of a page. Based on this idea, an alternative scheme uses the following equation to calculate the frequency factor $f(p)$, while keeping the other factors the same as in Equation 4.1. The approach is called *Subscription-GD*-2* or *SG2* in this thesis.

$$f(p) = s - a \qquad \text{- Equation 4.5}$$

Where

s : the number of subscriptions matching page p

a : the number of accesses of page p

Using either SG1 or SG2, the inflation parameter L is updated with the value of the page that is evicted last at the end of each replacement, as in GD*.

4.4 Algorithms for a More General Scenario

The algorithms presented in section 4.3 assume a strong correlation between users' pre-defined interest in subscriptions and their access patterns to events, which is hard to hold in practical applications. Many reasons may cause the mismatch between subscriptions and access patterns. For example, multiple subscriptions of a single user may match an event. However, the same user usually only requests the same page no more than once because duplicate requests to a same page can be intercepted in a browser-level cache. It is also very likely for a subscriber not to request the content of every event that matches the user's subscriptions. In any of above case, the total number of subscriptions that match an event exceeds that of accesses on the content.

More generally, a content server is not used for a particular publish/subscribe service; instead, it is shared by both pub-sub service providers and non-pub-sub service providers to serve end-users. The client of two types of services can be integrated via web browser through which users access information based on both notification services and general browsing as is done without a publish/subscribe environment. In this case, for a given event, users' accesses may be driven by notifications or not and hence the number of users' accesses may exceed that of subscriptions that match the event.

News delivery can be used to explain the more general scenario introduced above. Many news sites [CNN, MSNBC] provide notification service that is based on matching the contents to users' subscriptions. A user may not read all the pages that match the user's subscriptions (and of which they therefore receive notifications). Users may request news pages upon receiving the notifications or when they browse news sites independent of the notification service. The two types of accesses are called *notification-driven accesses* and *non-notification-driven accesses* or *general browsing* in this study. When pub-sub users and non-pub-sub users of a content server share interest in web content, subscription-based content delivery approaches benefit all users, whether they explicitly provided their interest in advance or not.

4.4.1 Our Previous Approaches Adapted for the New Scenario

When ignoring the type of access and viewing the two types of accesses in the same way, all approaches presented for pure publish/subscribe scenario except for Subscription-GD*-2 (SG2) are still applicable for the more general scenario.

The problem with SG2 is that Equation 4.5 assumes that the number of subscriptions that match a page is always larger than that of accesses of the page up to any point in time. The assumption holds when all requests are driven by notifications. However, if only some accesses are notification-driven, the result of Equation 4.5 may be negative, which is invalid for the evaluation function 4.1.A simple way to deal with the invalid case is to set all negative results to 0 when using Equation 4.5. In this way, when the number of accesses exceeds that of the subscriptions, SG2 ignores the frequency information and evaluates a page purely based on access recency. To distinguish from SG2, the above adapted version of SG2 is referred to as *Restrictive-Subscription-GD*-2* or *RSG2* in this thesis.

We also designed a set of new algorithms specifically for the more general scenario in which subscriptions do not reflect actual access patterns perfectly. The algorithms perform both publish-time and access-time placement, and can be further classified into three categories. The first algorithm uses the same replacement algorithm in publish-time module and access-time module. The evaluation function combines all information regarding a page's value as SG1, SG2 and RSG2. The second algorithm uses different evaluation functions in publish-time and access-time replacements. In the third class of algorithms, we designed approaches that partition a server's cache into two portions that are managed using independent modules.

4.4.2 Single Replacement Algorithm: Hybrid-User-GD* (HUG)

When users request contents via notifications and general browsing, the type of access, notification-driven or general browsing, becomes a new type of knowledge about user information needs. In practice, the information about the type of access can be obtained using protocols between end user's machines and content servers. For example, content servers can set up cookies in clients to track users' accesses driven by notifications.

The first approach we propose for the more general scenario applies the same placement algorithm at both publish-time and access-time, as do SR, SG1, SG2 or RSG2. Different from our previous approaches, the evaluation function in the new approach distinguishes the information associated with the two types of accesses. The new approach is called *Hybrid-User-GD** (*HUG*) in this thesis, since it considers the hybrid type of access and uses GD* framework.

HUG analyzes the frequency information of two types of accesses separately. For notification-driven accesses, the difference between the number of subscriptions matching a page and that of notification-based accesses up to the current time implies the amount of notification-driven accesses in the future (as in SG2 and SR). In contrast, the number of non-notification-based accesses in the future is derived from that in the past, according to the traditional LFU algorithms. The following equation estimates the number of future requests of a page using the sum of two factors each of which is associated with one type of access. The evaluation function of HUG is Equation 4.1 but using *f*(*p*) as defined by Equation 4.6.

$$f(p) = a_{NS} + (s - a_S) \qquad \text{- Equation 4.6}$$

Where
a_{NS} : number of accesses of page p that are not driven by notifications
s : number of subscriptions matching page p
a_S : number of accesses of page p that are driven by notifications

Interestingly, Equation 4.6 interprets the access frequency of two types of accesses in opposite ways: a high volume of past notification-driven accesses indicates low leftover value of a page for subscribers, while heavy general browsing implies a strong interest of web surfers in the page.

4.4.3 Dual Replacement Methods

The strategies that combine the subscription and the usage pattern into a single evaluation function are based on the assumptions on the relationship between the two patterns. When the correlation is not strong as in the more general scenario, one alternative is to use subscription information and access time information independently at publish time and at access time. Based on this idea, the publish-time placement module and the access-time replacement module run different replacement algorithms and different evaluation functions.

Using the new approach, every page is tagged with two values, one used in the replacement by the publish-time placement module and the other by the access-time placement module. At any time, the value of a page depends on whether the evaluation is for placing a newly published page or for an access-time replacement. We choose GD* to be applied in an access-time replacement and SUB in a publish-time placement. Since the two replacement algorithms use either access analysis or subscription information (neither uses both types of information), the two types of information are used separately in different modules in this way. This approach is referred to as *Dual-Methods* or *DM* in this thesis.

A potential problem with DM using independent functions to manage a single cache is that a page that is in common use will be replaced in a publish-time placement if the total number of subscriptions matching the page is not large enough. Or, on the other hand, a new page with a high future usage indicated by subscription matching will be replaced on a cache miss just because the page has not been referenced frequently up to the replacing time. This problem is due to the overlapping operations of the publish-time and the access-time placement modules in the same cache space.

4.4.4 Dual Caches Partitioned by Operation

To deal with the problem associated with *Dual-Methods*, the cache on a proxy can be divided into two portions that are used by the publish-time and the access-time modules independently. Under this scheme, the cache portion used by the publish-time module is called *Publish-Cache* or *PC* in this thesis, while the portion used by the access-time module is called *Access-Cache* or *AC*. Any page can stay in either *PC* or *AC* but not both at any time. The mechanism is denoted as *Dual-Caches Partitioned by Operation* or *DC-O* in our discussion.

Like *Dual-Methods*, *DC-O* runs GD* in the access-time module and SUB in the publish-time module. But unlike *DM*, each replacement algorithm runs only on the corresponding portion of a proxy cache. In this way, partitioning a server's cache provides a cleaner way than *Dual-Methods* for using subscription and access information separately.

We design different ways to handle a dual-cache as regard to whether the cache partition is static or adaptive. The following two subsections present one approach that uses a fixed partition and the schemes that partition a cache dynamically according to the publishing dynamics and access patterns.

4.4.4.1 Dual Caches Partitioned by Operation with Fixed Partition (DC-O-FP)

A simple way to handle a dual-cache is keeping a fixed partition on the storage. The approach is denoted as *Dual-Caches Partitioned by Operation with Fixed Partition* or *DC-O-FP* in this thesis. When a page is published and delivered to a server, the publish-time module runs SUB to store the page into *Publish Cache*. GD* is used to handle the replacement on *Access Cache* at cache misses.

Any portion of the dual cache can serve user requests. Therefore, the two cache portions are cooperative in locating a user's request. From a user's point of view, dual-caches work as a whole. In *DC-O-FP* when a page in *Publish Cache* is accessed for the first time, the page is moved (not copied) from *Publish Cache* to *Access Cache (AC*), meaning that the page should be henceforth evaluated based on the access pattern and be compared with other referenced pages in *AC* in the cache replacement algorithm. The locating algorithm with "Moving" operation in DC-O-FP is shown in Figure 4.4:

```
Page A is requested:
If A is in the publish cache PC
    Record a "hit"
    Move A from PC to the access cache AC
Else
    Run GD* on AC
Return page A
End
```

Figure 4.4 Locating algorithm in DC-O-FP

4.4.4.2 Dual Cache Partitioned by Operation with Adaptive Partition (DC-O-AP)

A fixed partition in a dual-cache lacks flexibility in using the cache storage according to the content publishing dynamics and the access dynamics. For example, when a page in the publishing cache *PC* is requested, the page is moved to the access cache *AC*, which possibly triggers a replacement in *AC* if *AC* is full. However, the storage in *PC* that was cleared by moving out the requested page is unused at least until the next new page is published. In such a case, a better strategy may be to reassign the storage of the requested page to the access cache. On the other hand, some storage in *AC* can be given to *PC* if there is no room to store a new page in *PC* **and** several old pages in *AC* have not been referenced for a while.

The approach that labels which partition each page belongs to according to the publishing and request patterns is called *Dual-Caches Partitioned by Operation with Adaptive Partition* or *DC-O-AP* in our discussion. Using *DC-O-AP*, when a page cannot be stored into *PC* based on *SUB* at publish time, the publish-time placement module checks the pages in *AC*. If some pages in *AC* have not been referenced since the last replacement in *AC*, these pages are assumed to be less important than the new page and thus become candidates for eviction. The storage of those pages is labeled as belonging to *PC* and is used to store the new pages. The placing algorithm of *DC-O-AP* is shown in Figure 4.5:

```
Page $P$ is published to the server:
Run SUB on the publish cache $PC$;
If SUB fails to store $P$
  $S \leftarrow$ pages in access cache $AC$ not accessed since the last replacement in $AC$
  If Size ($S$) $\geq$ Size ($P$)
    While the available storage in $PC <$ Size ($P$)
      $P_m$ s.t. $V_{P_m} = Min\{V_i, i \in S\}$
      Label the storage of $P_m$ as $PC$
    Store $P$ in $PC$
END
```

Figure 4.5 Placing algorithm of DC-O-AP

The "Moving operation" in *DC-O-FP* is replaced by labeling the storage of the page as *AC*, assuming the publishing frequency in *PC* is relatively low compared to the update frequency in *AC*. The locating algorithm of DC-O-AP is described using the pseudo-codes in Figure 4.6.

```
Page A is requested:
If A is in the publish cache PC
    Label the storage of A as belonging to the access cache AC
Else
    Run GD* on AC
END
```

Figure 4.6 Locating algorithm of DC-O-AP

Recall that the Dual-Methods for Single Cache (*DM*) strategy labels each page with two values and considers each value only in the corresponding module. In contrast, *DC-O-AP* labels each page with a 2-tuple (o, v) at any time, where o indicates the module that should process the page and v refers to the page's value under the corresponding operation. Both elements o and v are updated with time.

In *DC-O-AP*, the fraction of the storage assigned to each portion at a given point in time can be any value between 0 and 1. If either *PC* or *AC* dominates the storage on a server, publishing or caching consequently dominates the content distribution at that server. To avoid the possible imbalance in the influences of the two modules, a boundary should be set on the fraction of storage that can be assigned to each portion. A variant of *DC-O-AP* sets the upper boundary and the lower boundary on the fraction that can be *PC* or *AC* at any given time. We call this variant *Dual-Caches Partitioned by Operation with Limited Adaptive Partition* (*DC-O-LAP*). In *DC-O-LAP*, the re-partitioning in the placing and locating algorithms is performed only when the new partition makes the cache fraction used as any portion within the pre-defined range.

The *Dual Caches Partitioned by Operation with Adaptive Partition* approaches exploit the cache storage more intelligently than *Dual-Caches Partitioned by Operation with Fixed*

Partitioned (*DC-O-FP*). However, the approaches that partition a server's cache dynamically require more sophisticated cache management schemes than *DC-O-FP*. The complexity is mainly due to locating a page in the cache. Beside that, the fraction of each cache portion should be updated and kept for each server in *DC-O-LAP*.

4.4.5 Dual-Caches Partitioned by Type of Access: DC-TA

The key idea of the dual-caches approaches is exploiting different information sources in separate cache portions. The *Dual-Caches Partitioned by Operation* schemes (*DC-O*) use the subscription information only at publish-time and the access pattern only at access-time, hence *DC-O* divides a cache based on the type of operations, publishing and caching.

The type of access is another dimension along which to partition a cache space. Based on this idea a server's cache is divided into two equal portions that serve notification-driven and non-notification-driven accesses respectively. In our presentation this approach is called *Dual-Caches partitioned by Type of Access* (*DC-TA*). Using *DC-TA*, the cache portion that is consulted for notification-driven accesses is called the notification-based portion (*NP*). The other portion that serves general browsing is referred to as the browsing portion (*BP*).

NP is managed using *Subscription-GD*-2* since *SG2* captures the strong correlation between subscriptions and accesses best for subscribers according to the experimental analysis in chapter 6. *Browsing portion* works as a traditional cache using GD*. The page evaluation in *NP* is based on the subscription information and the accesses from subscribers of the pages, while the evaluation in *BP* relies only on the access pattern of general browsing.

NP and *BP* cooperate in locating requests of end users. If *BP* holds a page that is requested in a notification-driven access, the locating algorithm returns the page from *BP*. Similarly, a non-notification-driven request can be satisfied in *NP*. The cooperative locating between *NP* and *BP* is carried out in background and thus is transparent to end-users. From the point of view of users, the two portions seamlessly form a cache as a whole.

We assume that a publish-time placement for page *p* in *NP* is always completed before the first request of *p*. The access-time fetching occurs only when the page is not found in either *BP* or *NP*. Therefore, the cooperative locating in the two portions guarantees that no page resides in both portions at the same time.

However, *NP* and *BP* do not have to exchange information other than the existence of pages in each portion. In other words, each portion is read-only for the other portion. This design is suitable for an integrated caching system shared by multiple independent service providers. For example, one portion of a server's cache is assigned to a general-purpose content delivery service provider such as Akamai [Akamai], and another portion to a publish/subscribe service provider. The locating engine in *DC-TA* aggregates only the location information of pages in the portions, but leaves other privacy-sensitive information to each portion.

4.5 Summary of Strategies

Table 4.1 categorizes all the approaches discussed in this thesis as regard to when to place content at content servers and how the page is evaluated.

Table 4.1 Categorization of content distribution schemes

How When	Access	Subscription	Access + Subscription
Access-time	GD*		
Publish-time		SUB	
Access-time + Publish-time			SG1, SG2, SR, **RSG2, HUG** **DM,** **DC-O-FP, DC-O-AP, DC-O-LAP, DC-TA**

GD* is the baseline caching approach. SUB uses subscription information at publish-time only. We also propose several approaches that combine proactive publish-time placement and on-demand access-time placement using subscription information, access patterns, access type and other information together. The methods in bold are designed specifically for the more general scenario in which subscriptions may not match users' actual behaviors.

As discussed in the literature [Wang '99], the performance of the cache replacement algorithms depends highly on the traffic characteristics of accesses. It is worth pointing out that although we use *GD** as the framework in designing our publish-time and access-time combined schemes based on subscription as well as access patterns, our approaches can also be incorporated with other cache replacement algorithms.

Chapter 5

Simulator and Workloads for Evaluation of Algorithms

Because of the difficulty in obtaining real-world data about the publish/subscribe services assumed in this study, we evaluate the content delivery and caching approaches discussed in chapter 4 using a system simulator and the synthetic workloads that we build based on analysis and observations about the real-world data in the literature.

We chose news delivery to be the application scenario for our simulation because of its challenges. This is a challenging application since news publishing and reading have high temporal dynamics. In addition, news pages are of significant size, and many information objects embedded in media news (e.g., video, audio, and images) have large sizes. As indicated by Gadde et al. [Gadde '00], content distribution is more beneficial in applications that have a large number of popular objects that may have large sizes and high update frequencies. Henceforth, the news delivery that our workload generator mimics is challenging and thus can demonstrate the power of the content distribution strategies on which this thesis focuses.

5.1 Architecture of System Simulator

The system simulator for distributing the contents for a news site has the architecture as shown in Figure 5.1. The system simulator assumes a single publisher as a news site and a group of proxy servers, each of which connects to a set of users who are close to that server. To simplify our system simulator, we assume a centralized matching/routing engine. Nevertheless, the design of matching and routing engines for a whole publish-subscribe system does not affect the performance analysis on the content delivery algorithms in this study.

The input of the system simulator includes a publishing stream, a request sequence at each proxy-server, and the subscriptions collected from the end-users at each proxy-server. This input constitutes the synthetic workload we generate using our workload generator.

In our system simulator, the content servers accept subscriptions from the end users and upload the subscriptions to the matching/routing engine. The single publisher, for example, a news site, publishes events and the matching/routing engine notifies the servers whose users' subscriptions match the events. When an event is delivered to a server, the publish-time placing module of the server evaluates the content associated with the event and decides whether to store the content in the local cache. When a user requests a page that is not in a server's cache, the caching module of the server fetches the content from the publisher and runs the access-time replacement algorithm.

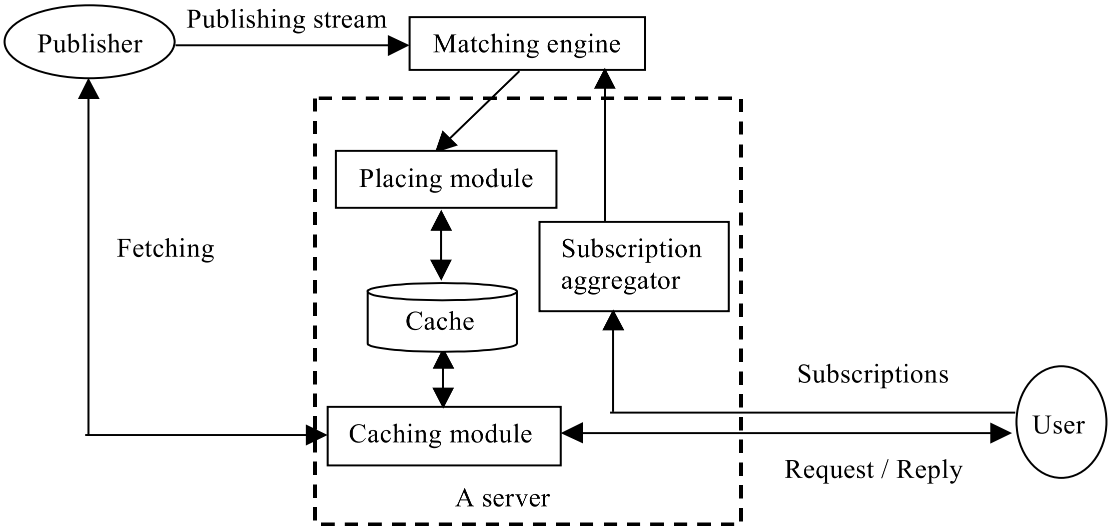


Figure 5.1. Architecture of the simulator

5.2 Workload Generation

Our study assumes that the fundamental users' request patterns in publish/subscribe applications are not different from that observed at general-purpose proxies and content sites. Similar to the request model presented by Breslau et al. [Breslau '99] and extended by Wolman et al. [Wolman SOSP '99] for proxy servers, our request model parameterizes the request rate, document rate of change, the total number of the information objects, and the popularity distribution for objects. Unlike the Wolman model, our request model simulates the number of proxy-servers rather than the population size of end-users because the proxy servers aggregate the requests and the subscriptions from the end-users and thus each proxy becomes a "virtual user" in our simulator. Another difference is that we consider the sizes of the objects and simulate a more realistic scenario in which every cache server has a limited storage capacity. Our model is also distinct from the previous models in that we incorporate subscription and matching information, whose distribution has not been addressed in the literature. Finally, our workload generator considers the content generation and access patterns at a whole publishing site, rather than that is observed at a forward proxy as in most trace analyses. This is because our study is about content delivery of whole contents from publishers such as a news site.

The subscriptions are information known in advance of the first access to each page, while the publishing stream and the request streams are temporal sequences. The following part of this section introduces the methods to generate the three input streams for a 7-day simulation. The publishing stream, request sequences and subscriptions are generated in that order, with every stream being built based on the streams built in the earlier steps. In generating the publishing and the request streams, most criteria and parameters are based on the analysis on the traffic [Padmanabhan '00] at the MSNBC site [MSNBC].

5.2.1 Generating Publishing Stream

For generating the publishing stream, we first need to decide the total number of pages that are published in 7 days. In the publishing stream, each page is associated with a publishing time and the page size. In the second step, the publishing time of each page should be determined based on the temporal patterns of content generation. Finally, the size of each page is decided.

5.2.1.1 Number of Pages and Publishing Time

According to the content dynamics observed at MSNBC [Padmanabhan ‘00], the total number of pages published in 7 days is about 30,000 with 6,000 of them being distinct pages. The other 24,000 pages are modified versions of about 2,400 out of the 6,000 distinct pages. Our publishing sequence consists of 30,147 pages in total.

The generation times of the pages are determined based on the observations of the modification intervals. At MSNBC, the cumulative distribution frequency of the modification intervals of all the 24,000 modifications has two distinct knees: the first is around 5% of all modifications and occurs at about 1 hour; another is around 95% and occurs at approximately 1 day [Padmanabhan ‘00]. This update rate is a feature of news sites and is much higher than the 14-day average update period of a popular object in the Wolman model [Wolman SOSP ‘99]. Given the above statistical observations, the modification intervals of the 2400 pages are built using a random number generator with 5% instances falling into the time period (0, 1 hour], 5% in [24 hours, 7 days), and 90% in (1 hour, 24 hours). Within each sector, the distribution of the modification intervals is uniform. We assume a fixed update interval for any page that is possibly modified multiple times.

Figure 5.2 demonstrates the cumulative distribution frequency (CDF) of the modification intervals in our publishing stream, with every modification on a page being counted. This distribution has two knees that are consistent with the observations in [Padmanabhan ‘00].

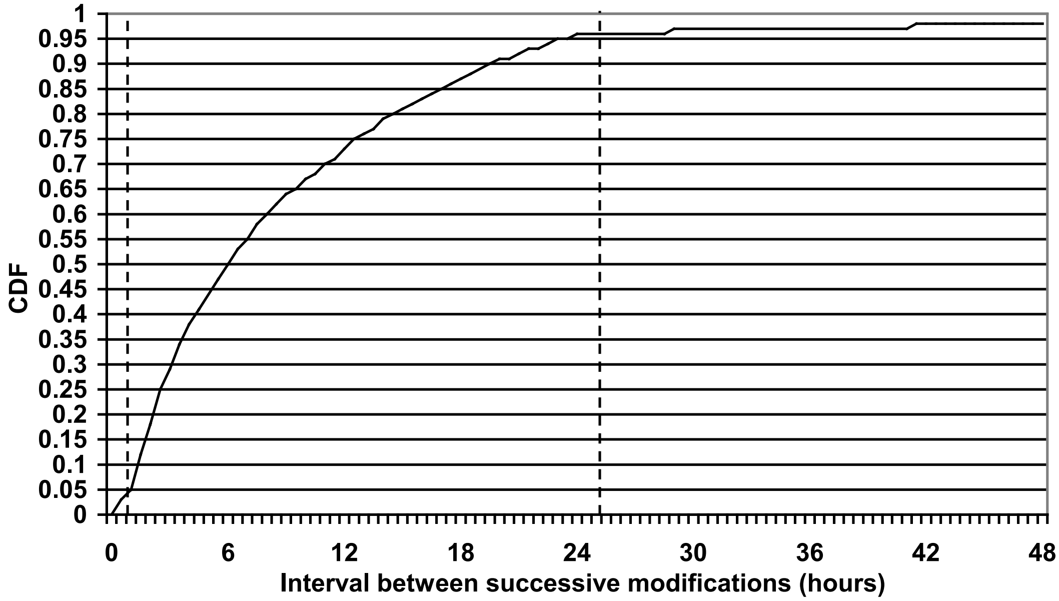


Figure 5.2 CDF of modification intervals for 24147 modifications(1 hr: 5% level, 24 hrs: 95% level)

The first publishing time of any of the 6,000 distinct pages is randomly chosen from the period (0, 7 days), and the generation times of the 24,000 modified versions are decided based on the modification intervals and the generation times of the first versions. Figure 5.3 shows the generation frequency of pages with each version of a page (new or modified) being counted as a separate page.

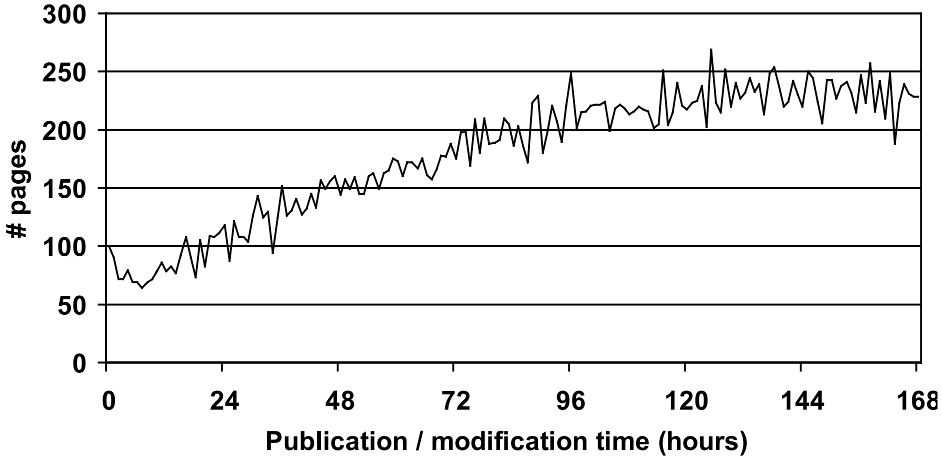


Figure 5.3 Publishing frequency of 30147 pages in 7 days

The modifications to a news page stop after some time, and the rate of publishing is the sum of new page generations and old page updates in figure 5.3. Hence the publishing frequency increases with time in the "warm-up" period and then becomes stable later. In the following discussion, "page" refers to any page in the 30147 published pages, including both the original pages and the modification versions.

5.2.1.2 Page Size and Cacheability

The sizes of the pages are generated using a log-normal distribution according to the observations in literature [Barford '98][1]. As suggested in a previous study [Venkataramani '01 TR], we assume no correlation between the size and the popularity of a page.

All the pages are assumed to be cacheable in our simulation, which is an optimistic assumption for caching technology. In reality, some pages are tagged with "non-cacheable by proxy caches" due to security concerns. This is not a barrier for a content delivery network (*CDN*) since a CDN is the surrogate of publishers rather than end users. Therefore, in practice, more pages are cacheable for a *CDN* with proactive placement module than for a client-side proxy cache. The assumption that all the pages are cacheable biases our experimental results toward the caching-only approach.

5.2.2 Generating Request Stream

After deciding the publishing stream, the second step in building our workloads is generating the request trace at each content server. Information about each request consists of three parts: the requested entity (page ID), the issuing time, and the server that issues the request. This section first describes the approach to scale down the simulation size by randomly choosing 100

[1] $p(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-(\ln x - \mu)^2 / 2\sigma^2}, \mu = 9.357; \sigma = 1.318$

content servers into our simulation. After that, the total number of requests and the time of each access to each page are decided. The last step in request generation is splitting the global request stream to the access streams per content server.

5.2.2.1 Scaling Down the Number of Requests

As observed in the literature [Padmanabhan ‘00], there are about 25 million requests to the MSNBC site every day, so there should be about 175 million requests to the publishing site in a 7-days trace to mimic a busy site like MSNBC.

To scale down our simulation, we consider a request workload as a part of the global trace only. We assume that 100 servers are randomly chosen from all the servers sending requests to the MSNBC site from all over the world. Therefore the behavior of each of the 100 servers is representative of that of a proxy server that serves a domain such as an institute. Under this hypothesis, our request generator uses the global request pattern as reported in [Padmanabhan ‘00] to shape the aggregated request pattern from the 100 servers. According to the definition of institutional domains in [Padmanabhan ‘00], the analysis of 5-day traces reveals that at least several hundreds of thousands of domains request MSNBC, hence we assume the 100 servers issue about 1/1000 of all the requests to the site. In this way, the request rate in our 7-day traces is scaled down to around 195,000.

Instead of choosing a fraction of the global requests to a publishing site as we did, an alternative approach is to distribute the global requests to a site in the traces on these 100 servers. This approach is actually to aggregate the requests from multiple servers into a trace on one server. However, as pointed out in earlier studies [Gadde ‘00, Wolman SOSP ‘99], the hit ratio grows quickly with the population size served by each proxy. This implies that the behavior of a server with aggregated accesses from several actual proxies cannot represent that of a single proxy in reality.

One of the interesting results in [Padmanabhan '00] is that the page popularity distribution at the news site follows Zipf's Law[2] with parameter α taking the high value of 1.5. This observation is consistent with the observation that the set of popular pages is smaller on a globally popular site than not popular sites [Gwertzman '97]. Our request generator uses Zipf's Law with α of 1.5 to model the popularity distribution of the 30147 pages published in our simulation. The popularity rank of every page is randomly assigned, assuming the rank is independent of the publishing time. The total number of references to a page is determined using the popularity rank of the page, the popularity distribution, and the total number of requests in the 7-day trace.

5.2.2.2 Deciding Request Times

Given the total number of requests to each page, one needs to determine the issuing time of each request. Our request generator determines the reference times based on the correlation between a page's age and the probability that the page is accessed. According to the observations in [Padmanabhan '00], most news pages are requested when they are fresh, but popular pages are still referenced even if they have been generated for a long time. It is also observed in an independent study that the probability of referencing an object *t* units of time after the last reference of the object is roughly proportional to 1 / *t* [Breslau '99].

Based on the above qualitative observations, the reference model of a page is related to the page's popularity. We cluster the pages according to their popularity and define a reference model to each class. One way to cluster pages according to popularity is to make the request rate drops about one order of magnitude from one class to the next. In this way all the pages are grouped into four classes according to their popularity: the pages whose ranks are between 0

[2] $R_i = {}^{1}\!/_{i^{\alpha}}$, the request rate on the page with rank *i*

and 20, those between 20 and 350, those between 350 and 1400, and those above 1400. A page's lifetime is the period between the page's generation time and the time of the next modification to the page, or, if it is not modified, the period between its generation time and the end of the seventh day. A page's relative age is represented as one of the four quartiles that divide a page's lifetime equally. For a given class, the probability of a page being requested is inversely correlated to the page's relative age as shown in Figure 5.4. Within each quartile in lifetime for a given class, we assume the distribution of the access times is uniform.

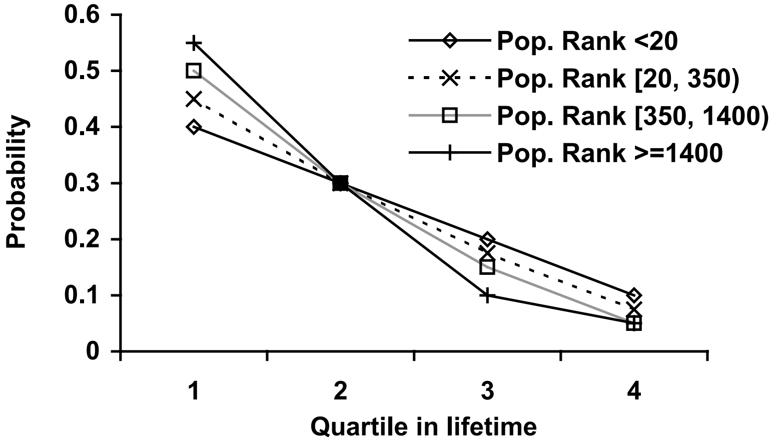


Figure 5.4 Probability of access w.r.t relative age

The access times of a page are distributed in the 7-day window based on the probability distributions in Figure 5.4 and the publishing time of the page. The resulting distribution of all the requests is shown in Figure 5.5. Although not generated by design, the spikes in the resulting request stream from the 100 servers demonstrate the *burstiness* in the overall traffic to a news site, which reflects an important characteristic in web traffic.

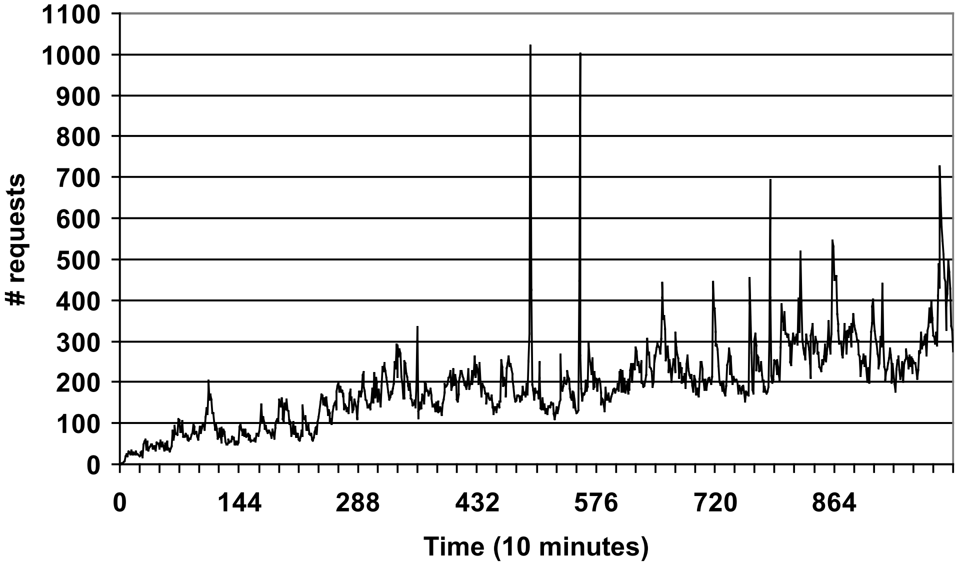


Figure 5.5. Number of requests issued at different times

5.2.2.3 Splitting Requests by Server

The traces per server are extracted from the aggregated request sequence generated so far. As observed in the literature [Padmanabhan '00, Wolman USITS '99], the frequently referenced pages are usually globally popular and are accessed by more organizations. This observation implies a positive correlation between the number of servers requesting a page and the page's popularity. This correlation is the basis of our server assignment.

As the first step, the server assignment algorithm decides the maximum number of servers requesting a given page in a day as a function of the page's popularity using Equation 5.1. To avoid a rigid correlation between the popularity and the number of servers requesting a page, S_i is scaled by 10% up or down in a random way for each page after applying Equation 5.1.

$$S_i = 100 \cdot \left(P_i \middle/ P_{max} \right)^{0.5} \qquad \text{- Equation 5.1}$$

Where
P_i : the popularity of page *i*
P_{max} : the maximum popularity of all the pages

For the first time that a page is requested, S_i servers are randomly chosen from the 100 servers to make up a pool of potential servers. After that, every request to that page in that day is randomly assigned to one of the S_i servers. The random assignment implies that not every server in the pool may be used.

As observed in [Padmanabhan '00], the server group requesting a page in one day and that in the next day overlap. Assuming the overlapping ratio is 60%, 40% of the candidate servers for a page in one day are replaced by the servers that are not in the current pool when generating the candidate servers for the page in the second day. The requests to a page may span several days. Figure 5.6 shows the maximum number of servers requesting each of the 2000 most popular pages in a day during the 7-day simulation.

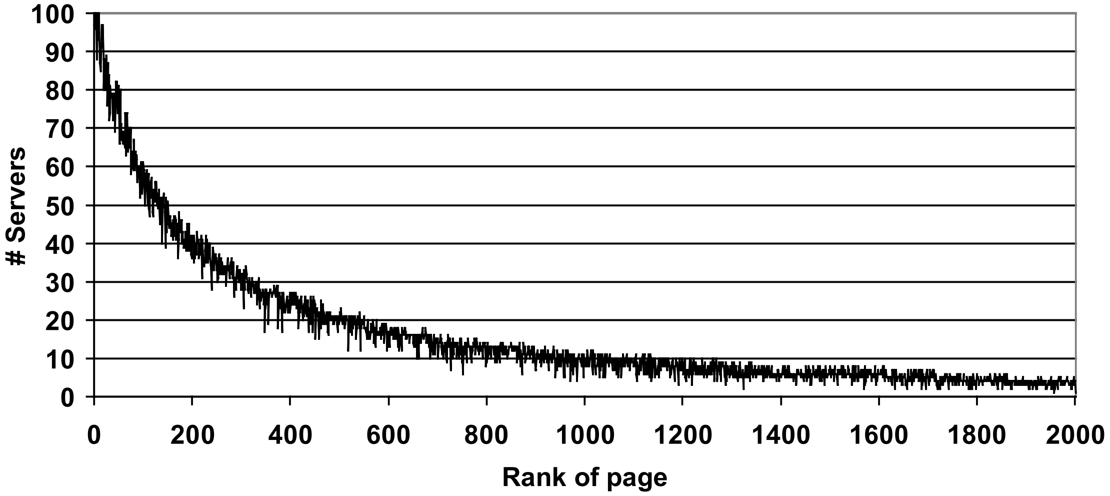


Figure 5.6. Number of servers visiting a page in a day

The first time miss ratio (*FTM*) in the requests at a server, the percentage of accesses that are to the objects that have not been accessed before by users at the server, gives an upper bound for the hit ratio of access-only based caching algorithms for the server. The first time miss ratio is determined by the distribution of pages' popularity and the number of requests. Given a number of requests, the more uniform a page popularity distribution, the lower the

FTM. After the server assignment, the average *FTM* at a server in our request traces is 33%, which is lower than the 40% usually observed on a proxy server in the literature. The low *FTM* in our trace is due to the relatively large α in Zipf's Law that models the popularity distribution of the news pages. Figure 5.7 shows the distribution of *FTM* at the 100 servers. The high variance of *FTM* among the servers is consistent with our assumption that the 100 servers are randomly chosen and thus have different reference patterns.

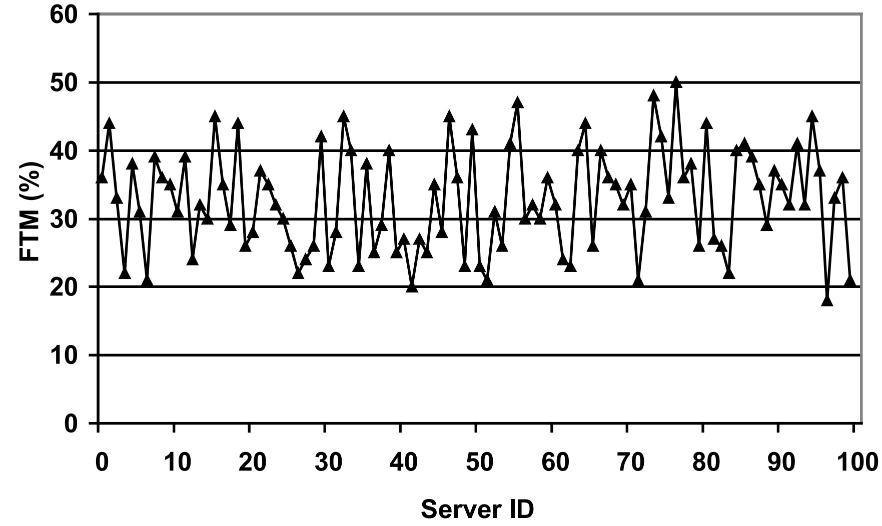


Figure 5.7 First time miss rate at the 100 servers

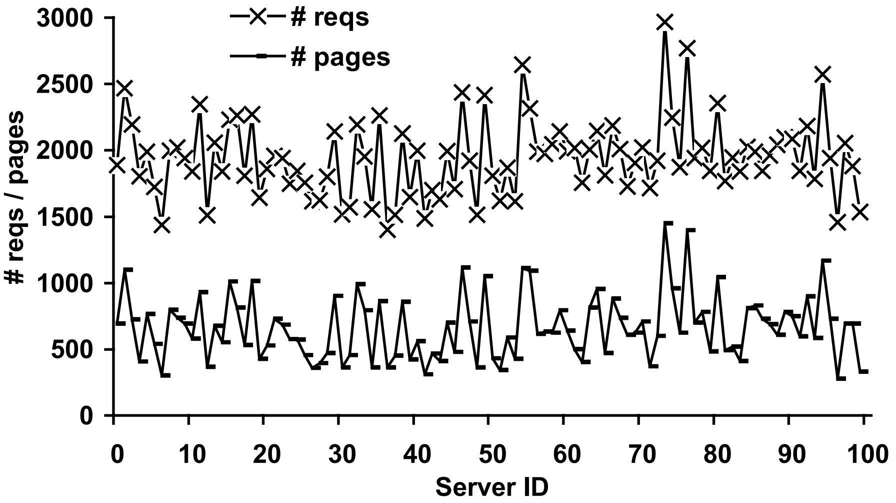


Figure 5.8 Total number of requests and the number of distinct pages requested at the 100 servers

In Figure 5.8, the total number of requests at a server is positively correlated to the number of the distinct pages requested from a server. Reading Figures 5.7 and 5.8 together, a positive correlation can be found between *FTM* and the number of distinct pages requested at a server. This observation matches the assumption that the miss stream is divided among *CDN* servers in proportion to the number of objects assigned to each server in the study of [Gadde '00].

5.2.2.4 Generating Request Traces with $\alpha = 1.0$

While news delivery is the focus of our validation study, the performances of our content delivery strategies for more general scenarios are also of interest. As a comparison, another request trace is built using a more popular α value of 1.0 in the popularity distribution that follows Zipf's Law. Except for α, all the other criteria and parameters for generating requests are the same as before. This trace has an average *FTM* of 50% with a distribution as shown in Figure 5.9. In the rest of this thesis, the set of request traces and subscription information built using $\alpha = 1.5$ is called NEWS, and that using $\alpha = 1.0$ is called ALTERNATIVE.

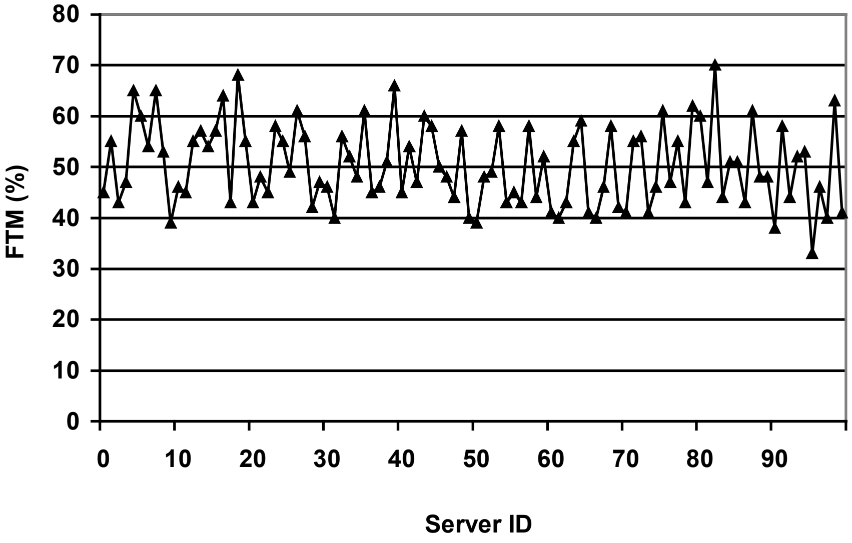


Figure 5.9. First time miss rate at the 100 servers (α = 1.0)

5.2.3 Generating Subscription Information

In addition to the publishing and the request traces, the third trace we used is the sequence of end-user subscriptions. In this study, the subscriptions are assumed to be static information that is known in advance. This assumption is reasonable since the update rate of user subscription is usually much lower than the rate of user access. The only subscription information used in our algorithms is the number of subscriptions matching each page at each content server, which can be inferred from the total number of requests to the page from the server.

Equation 5.2 determines the number of subscriptions that match a page *i* at a server *j* based on the total number of requests to *i* at *j* and the correlation between the subscriptions and the accesses. The correlation is determined by two factors: the ratio of the number of notification-driven accesses to that of all accesses for the page at the server, and the ratio of the number of the notification-driven accesses to that of the subscriptions for a page. The first parameter $\{F_{i,j}\}$ models the composition of two types of accesses, while the second one, $\{SQ_{i,j}\}$, quantifies the probability for a subscriber to read a page that matches the user's subscriptions, or the quality/accuracy of a user's subscriptions.

$$SF_{i,j} = \frac{P_{i.j} \cdot F_{i,j}}{SQ_{i,j}} \qquad \text{- Equation 5.2}$$

Where

$SF_{i,j}$: number of subscriptions matching page *i* at server j

$P_{i,j}$: number of requests to page *i* at server j

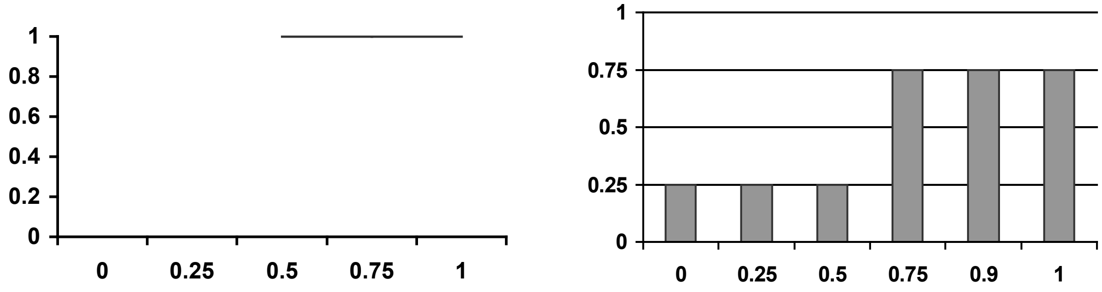
$F_{i,j}$: fraction of notification - driven requests to page *i* at server j

$SQ_{i,j}$: probability for subscribers to read page *i* at server j

For a given user model, we use the global parameter F to describe the estimate of $\{F_{i,j}\}$, the fraction of notification-driven accesses for all page/server pairs. The global parameter SQ stands for global subscription quality, and it is the estimate of $\{SQ_{i,j}\}$, the probability for a subscriber to actually access a page that matches the user's stated interest (i.e., for which the subscriber receives a notification).

By definition pure pub-sub applications have F of 1, which means all user accesses are driven by notifications. The global subscription quality *SQ* being 1 is the special case that every user will in fact access any page matching the user's interest after the user is notified.

Given a distribution for the $\{F_{i,j}\}$ or $\{SQ_{i,j}\}$ and a global parameter *F* or *SQ*, one can generate $\{F_{i,j}\}$ and $\{SQ_{i,j}\}$, respectively, for all pages and servers, and hence build the subscription information using Equation 5.2. Given the lack of evidence in the literature about the distributions of $\{F_{i,j}\}$ and $\{SQ_{i,j}\}$ as defined, we explore three different distributions to model $\{F_{i,j}\}$ and $\{SQ_{i,j}\}$. The first distribution is a step-wise function with the disjoint point at the estimate value of the sequence. The second distribution is a uniform distribution. The last one is a Gaussian-like distribution. All three distributions have a domain of [0, 1]. Figure 5.10 shows the uniform function and the step-wise function for a global parameter of 0.75.



(a) Uniform distribution **(b) Step function**

Figure 5.10 Functions of $\{F_{i,j}\}$ and $\{SQ_{i,j}\}$ for a given global parameter F or SQ of 0.75

Since the experiments using the three distributions yield similar results, we focus our discussion on the results using a Gaussian-like distribution to model $\{F_{i,j}\}$ and $\{SQ_{i,j}\}$.

5.2.4 Tagging Accesses as Notification-Driven or Not

Since the two algorithms HUG and DC-TA require knowledge about the type of access, each access in our request streams should be identified as either notification-driven or non-notification-driven. Given $P_{i,j}$ and $F_{i,j}$ as defined in Equation 5.2, $P_{i,j} \cdot F_{i,j}$ accesses are randomly chosen from the access sequence to page *i* at server *j*. The chosen accesses are tagged as notification-driven, while others are marked as general browsing accesses.

Chapter 6

Experimental Results on Subscription-enhanced Content Delivery

We compared the caching and content delivery mechanism presented in chapter 4 against each other by using the workloads and the simulator discussed in chapter 5. The experimental results are demonstrated in two sections. In the first section of this chapter, we analyze the algorithms' performance for the cases in which all users' requests are driven by the notification messages but a notification may or may not cause a user access to the associated content. The second section in this chapter demonstrates the performances of different approaches in the more general scenarios in which not only notifications but general browsing may also drive users' accesses of contents. Table 6.1 reviews the key characteristics of our approaches.

Table 6.1 Review of the characteristics of important algorithms

Name of Method	Key characteristics
Dreedy-Dual* (GD*)	an caching approach
Subscription-based placement (SUB)	*f* = *s*
Subscription-GD*-1 (SG1)	*f* = *s* + *a*, with recency analysis
Subscription-GD*-2 (SG2)	*f* = *s* – *a*, with recency analysis
Subscription-Request (SR)	*f* = *s* – *a*, without recency analysis
Hybrid-User-GD* (HUG)	$f(p) = a_{NS} + (s - a_S)$, with recency analysis
Dual-Method (DM)	SUB for publish time and GD* for access time, a single cache
Dual-Caches Partitioned by Operation (DC-O)	SUB for publish time and GD* for access time, dual caches
Dual-Caches Partitioned by Type of Access (DC-TA)	SG2 for notification-driven accesses and GD* for general browsing, dual caches

6.1 Comparing Algorithms When All Accesses are Driven by Notifications (F = 1)

The experiments in this section demonstrate the performances of different approaches when all accesses to contents are driven by notifications (F = 1). Without specification on subscription quality (SQ), the results are for the cases when each notification causes one access (SQ = 1).

6.1.1 Metrics and Experimental Setup

Our main motivation for designing a content delivery scheme is to reduce the response time perceived by the end users. The communication latency between users and a local proxy-server is usually much smaller than that between users and the publisher. Consequently, a high hit ratio in a local server generally means a smaller response time. We use three key metrics to evaluate the approaches. They are the *global hit ratio* (*H*) on the 100 servers, the workload at the publishing site, and the network traffic. Global hit ratio is defined in Equation 6.1.

$$H = \frac{\sum_{i=1}^{100} H_i}{\sum_{i=1}^{100} R_i} \qquad \text{- Equation 6.1}$$

Where

H_i : the number of hits on server i

R_i : the number of requests on server i

A desirable approach should improve *H* without introducing a big overhead into the network traffic between the publishing site and proxy servers. Snapshots of the network traffic are highly dynamic and depend on many factors. To simplify the evaluation, the network traffic is measured using the total amount of contents triggered by either the publishers or the servers within the whole network at different times in our experiments. The amount of traffic is measured using the number of pages as well as the total number of bytes.

Our simulation-based experiments model a scenario in which the storage capacity of each cache is limited. The storage capacity is set per server, assuming each server can be configured independently according to its workload. The experimental results on the chief performance metric *H* of each algorithm are demonstrated, but the major discussions are around the performances of the methods that are tested under three different settings for cache capacity: 1% of the total number of unique bytes requested by each server in the whole simulation, 5%, and 10%, according to typical cache settings in the literature.

The experimental results presented in this thesis can be extended for the cases in which multiple publishers generate events with associated contents. If the content generation is independent on multiple publishing sites and the access patterns on these sites have similar characteristics as discussed in chapter 5, the total number of documents and that of requests will scale up with the number of publishers. Of course, if we want to use the same cache capacity settings in percentage as in this thesis, the actual cache size of each content server required for delivering contents from multiple publishers should be larger than that for a single publisher.

In this section, we first examine the influence of the parameter β in equation 4.1. Then we compare the *Dual-Caches partitioned by Operation* (*DC-O*) approaches and identify the best algorithm in *DC-O* family. By comparing the best DC-O approach and Dual-Method (DM) algorithm, we further determine the winner between the two algorithms. After narrowing down the algorithms for further study, we analyze the major algorithms' performances for the trace NEWS. Using this trace, we first measure the global hit ratio of the major algorithms under different cache capacity settings; then focusing on the cache capacity 5%, we analyze the algorithms' hit ratio under different subscription quality SQ (the probability for a notification to drive an access). For the most important algorithms, we compare them regarding the temporal hit ratio, the workload at the original publishing site, and the network traffic. For the trace ALTERNATIVE, we repeat the major analyses on the algorithms.

6.1.2 Influence of β

According to experimental results [Jin '01], the parameter β (see equation 4.1 repeated below) that balances the long-term popularity and the short-term temporal correlation in GD* should be set differently for optimal performances from trace to trace. On the other hand, when β is learned on-line from the past accesses seen at different times, β is quite stable for a given trace. To decide the suitable value of β, GD* and the two GD*-based approaches subscription-GD*-1 (SG1) and subscrption-GD*-2 (SG2) are evaluated using different settings of β.

$$V(p) = L + \left(\frac{f(p) \cdot c(p)}{s(p)} \right)^{1/\beta} \qquad \text{- Equation 4.1}$$

Where

L: inflation value to capture the access recency

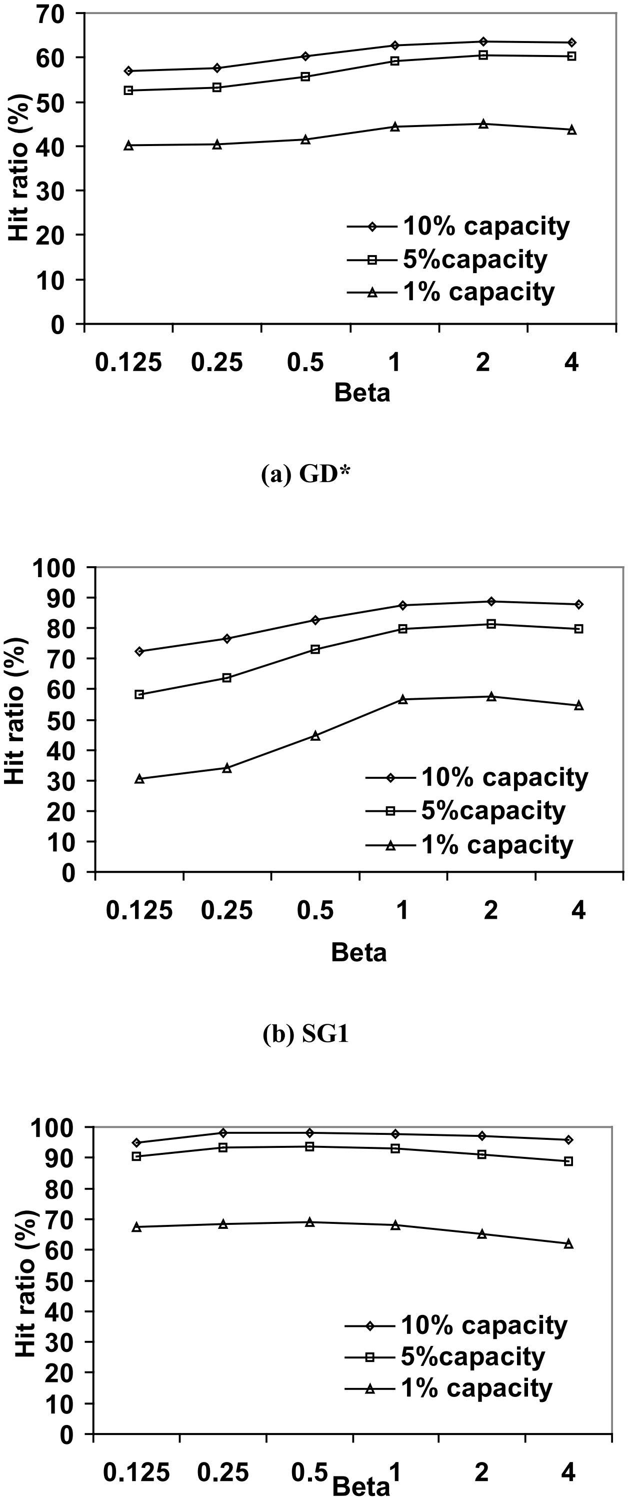
$f(p)$: number of accesses on the page

$c(p)$: cost to fetch a page from the publisher

$s(p)$: page size

β: balance factor of popularity and temporal correlation

Figure 6.1 shows the performance of the three methods for the trace NEWS. Figure 6.1(a) shows that GD* achieves the highest hit ratio for this trace when β is 2. This optimum value of β for our request trace is larger than 0.5 that was calculated for the traces in [Jin '01]. A larger β means a higher importance of access recency than popularity in the evaluation function of Equation 4.1. We conjecture that the difference results from the higher temporal correlation in the requests for news stories. Figure 6.1(b) demonstrates the hit ratios of SG1 when the value of β varies from 0.125 to 4. As with GD*, the value 2 is the best choice for β when using SG1. On the other hand, the hit ratio of SG2 is maximized when β is 0.5, as shown in Figure 6.1(c). The above observations about the best value of β for the three methods hold for all three cache capacity settings we used in the experiments.



(a) GD*

(b) SG1

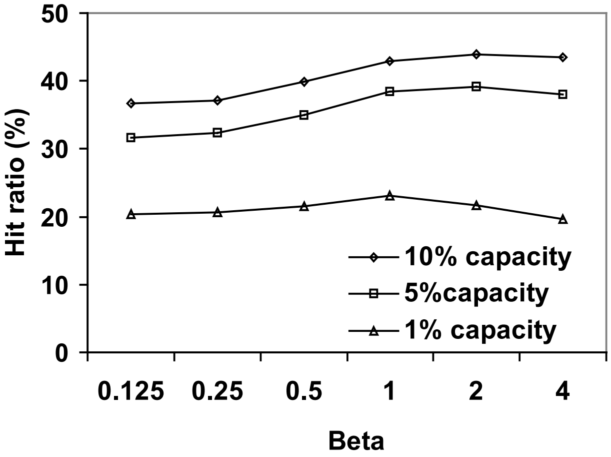
(c) SG2

Figure 6.1 Hit ratios of GD*, SG1 and SG2 with different β (NEWS)

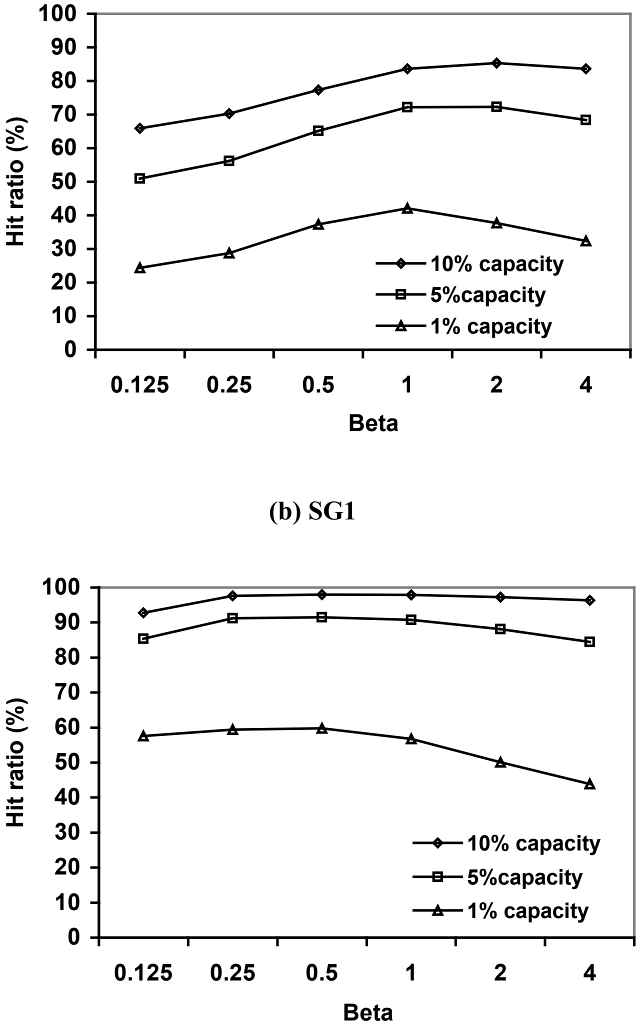
Among the three methods, SG1 is most sensitive to β. This is due to the large base value of the exponential in the evaluation function of SG1 (see Equation 4.4). However, the sensitivity of SG1 to β is very low if β is not less than 1. In contrast, SG2 is least sensitive to β because the base value for SG2 (see Equation 4.5) is usually smaller than that in GD* and SG1.

For the trace ALTERNATIVE that has the popularity distribution of pages following Zipf's Law with α = 1.0, the hit ratios of GD*, SG1 and SG2 with different values of β are shown in the Figures 6.2. In general, the performance of each algorithm changes with β in similar ways as for NEWS, except that the best result using GD* or SG1 is achieved when β = 1 rather than 2 when the cache capacity setting is very low (1%). For SG2, the hit ratio is still quite stable with β when the cache capacity setting is 5% and 10%, but its hit ratio is affected dramatically when the capacity setting is 1%.

Assuming a suitable value of β can be learned for each method, in the following experiments, the value of β is set to maximize the hit ratio for each algorithm over the trace. Namely, β is 2 in GD* and SG1 for the trace NEWS; for ALTERNATIVE, β is 2 in GD* and SG1 when the capacity setting is 5% or 10% and 1 for 1%. β is always 0.5 in SG2.



(a) GD*



(b) SG1

(c) SG2

Figure 6.2 Hit ratios of GD*, SG1 and SG2 with different β (ALTERNATIVE)

6.1.3 Best of Dual-Methods and Dual-Caches Partitioned by Operation

Dual-Methods (*DM*) and Dual-Caches Partitioned by Operation (*DC-O*) apply the subscription information and access information separately in publish-time placement and access-time placement. *DM* carries out the placement algorithms on a single cache, while DC-O performs the algorithms on a dual-cache. The experiments in this section compare the approaches in these two categories and identify the best approach in Dual* family (including *DM* and *DC-O*).

6.1.3.1 Analyzing the Influence of Partition in DC-FP

Figure 6.3 demonstrates the hit ratios of Dual-Caches Partitioned by Operation with Fixed Partition (*DC-O-FP*) for different traces and cache sizes with different fraction of storage given to the publish-time cache, shown as the values in the legend. There is no obvious difference when assigning 20%, 25% or 50% of total storage to the publish-time cache at a server, but neither giving too much storage (75%) nor too little storage (10%) to the publish-time cache is a good strategy for this trace. The results for ALTERNATIVE are similar to that for NEWS.

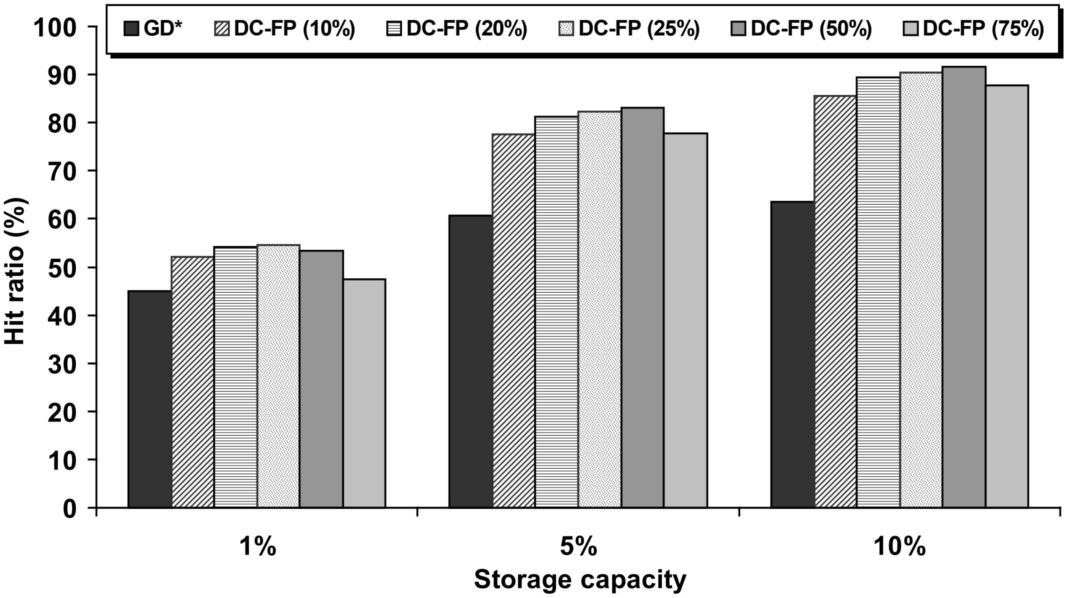


Figure 6.3 Hit ratio of DC-O-FP vs. partitions (NEWS)

6.1.3.2 Comparing DC-O with Fixed Partition and Adaptive Partition

Figure 6.4 compares DC-O-FP, Dual-Caches Partitioned by Operation with Adaptive Partition (DC-O-AP), and Dual-Caches Partitioned by Operation with Limited Adaptive Partition (DC-O-LAP). DC-O-FP uses a 50%-50% partition, while DC-O-AP starts from a 50%-50% partition but adjusts the partition based on the publishing pattern and the reference pattern. DC-O-LAP limits the re-partition in DC-O-AP by bounding the fraction of storage of each cache portion between 25% and 75%.

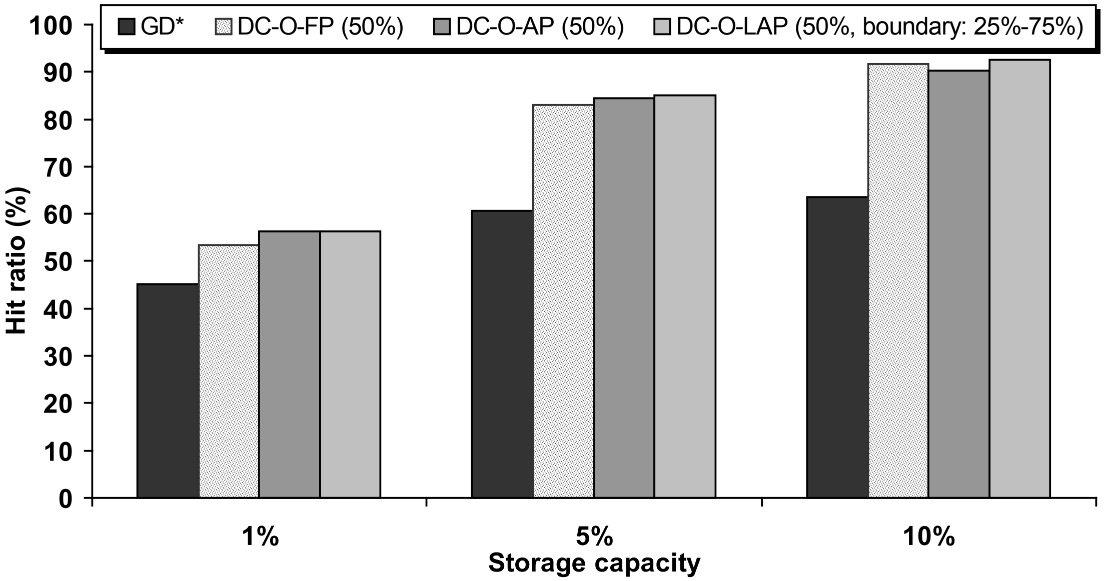


Figure 6.4 Hit ratios of Dual-Caches Partitioned by Operation with different cache sizes (NEWS)

Adding constraints to the cache partition (i.e. DC-O-LAP) helps to improve the hit ratio of DC-O-AP and make it uniformly better than DC-O-FP, especially when SQ < 1 as shown in Figure 6.5. However, the marginal improvement using adaptive partition is very small when all users' accesses are driven by notifications and every notification is assumed to result in one user request. A similar result is observed for the trace ALTERNATIVE.

We conjecture that the small improvement using the more intelligent approach DC-O-AP results from the high publishing frequency and high re-access frequency in our traces. Recall that our motivation to adaptively decide the dual-cache partition is to expand the publish-time cache when the pages in the access cache are not used in recent references and on the other hand, to expand the access cache if the publishing stream comes slowly. In our simulator, the publishing frequency is as high as about 3 pages per minute, and the rate of repeated references of a page at a server is high (the repeated requests constitutes about 67% of all requests in

NEWS trace and 50% in ALTERNATIVE trace). The high rate of repeated references is a result of both large α in the popularity distribution and the strong correlation between a page's recency and the access probability (see Figure 5.4). Therefore, the power of DC-O-LAP is not obvious for our traces that have both high publishing rate and re-access rate.

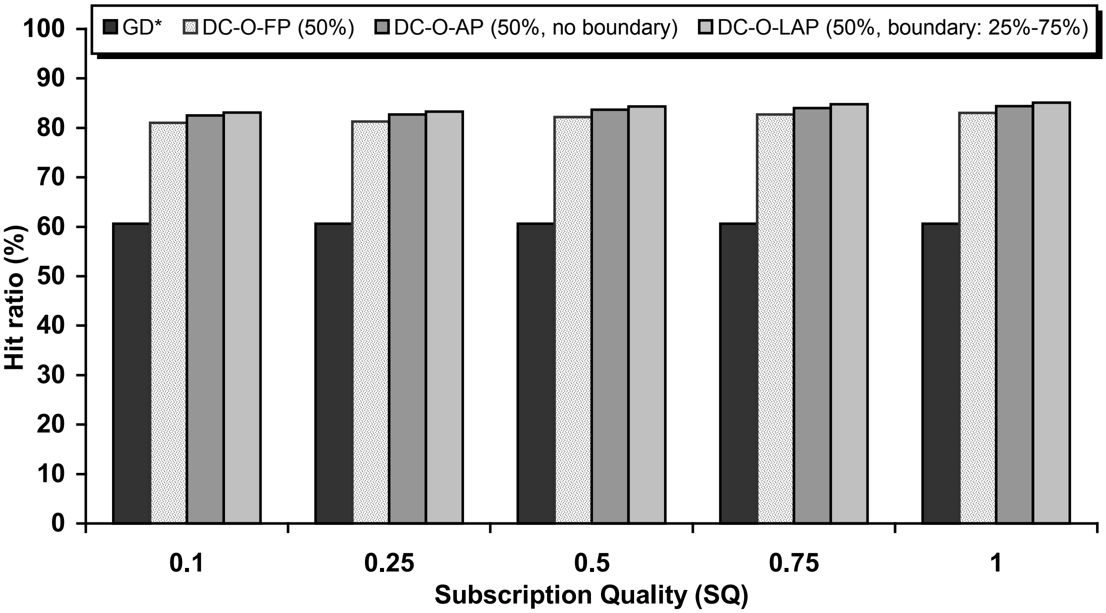
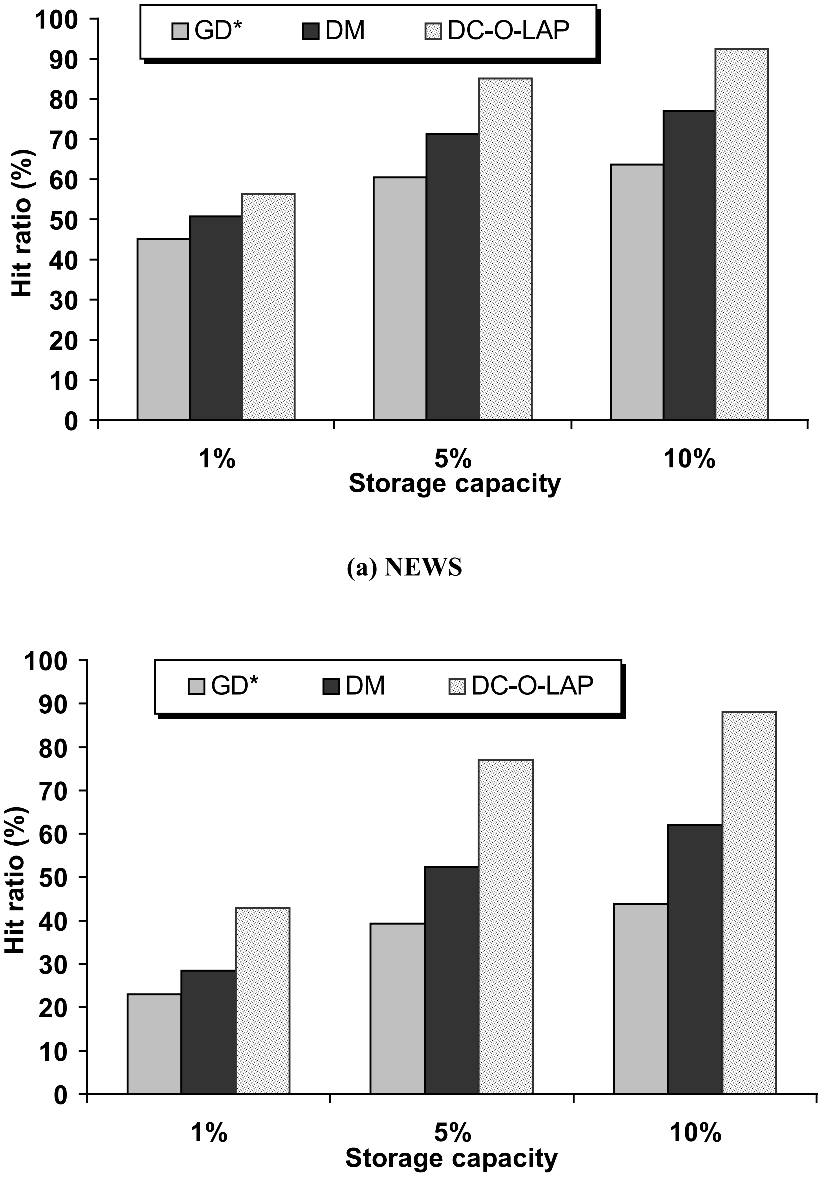


Figure 6.5 Hit ratios of Dual-Caches Partitioned by Operation with different SQ (cache size = 5%, NEWS)

6.1.3.3 Comparing DM and DC-O-LAP

Figures 6.6 compares DM and the best Dual-Caches Partitioned by Operation approach, DC-O-LAP with boundaries set at 25% and 75% of the cache, for two traces. Under different cache capacity settings, DM consistently performs worse than DC-O-LAP. This is especially true when the popularity of pages is more uniform.



(a) NEWS

(b) ALTERNATIVE

Figure 6.6 Hit ratios of three Dual* approaches

6.1.3.4 Summarizing Dual Methods*

The above comparison results are for the case that the subscription quality (SQ) is 1, but the ranks of the algorithms hold when SQ varies from 1 to 0.25. All the Dual* approaches have better hit ratio than GD* as shown in the Figures 6.3 to 6.6, but DC-O-LAP outperforms DM and other DC-O approaches in all the cases. Therefore, DC-O-LAP is chosen as the representative of the Dual* family in the following comparison experiments.

6.1.4 Performance of Major Algorithms for NEWS

Cache capacity and subscription quality (SQ, see Equation 5.2) are two dimensions to analyze the algorithms' performance. This section first compares the global hit ratio of the approaches under different cache sizes with a fixed subscription quality (SQ) as 1, then under different assumptions about *SQ* with a fixed cache size as 5%. Assuming SQ = 1 and cache size is 5%, the temporal hit ratios, the workload at the publishing site and the network traffic of the major algorithms are demonstrated and analyzed in that order.

6.1.4.1 Comparison of the Approaches Under Different Cache Sizes

Figure 6.7 shows the hit ratios of the algorithms under cache sizes from 1% to 100% of the total number of unique bytes requested at each content server for the NEWS trace in the ideal case that the subscription information perfectly reflects users' request frequencies in the future (*SQ* = 1). Given the first time miss ratio for NEWS traces is around 34.2%, the optimum hit ratio of GD* is 65.8%. All the subscription-based approaches are able to achieve 100% hit ratio when the cache size is large because the algorithms proactively place contents before any request to the contents. Under any cache size, all our subscription-enhanced approaches outperform GD* except for SUB at cache size = 1%. Figure 6.7 also shows that the hit ratio using any algorithm changes dramatically from 1% cache size level to 10% cache size, hence we focus on the results at the three capacity levels, 1%, 5% and 10%, in the following analyses on the algorithms' performance for the NEWS trace.

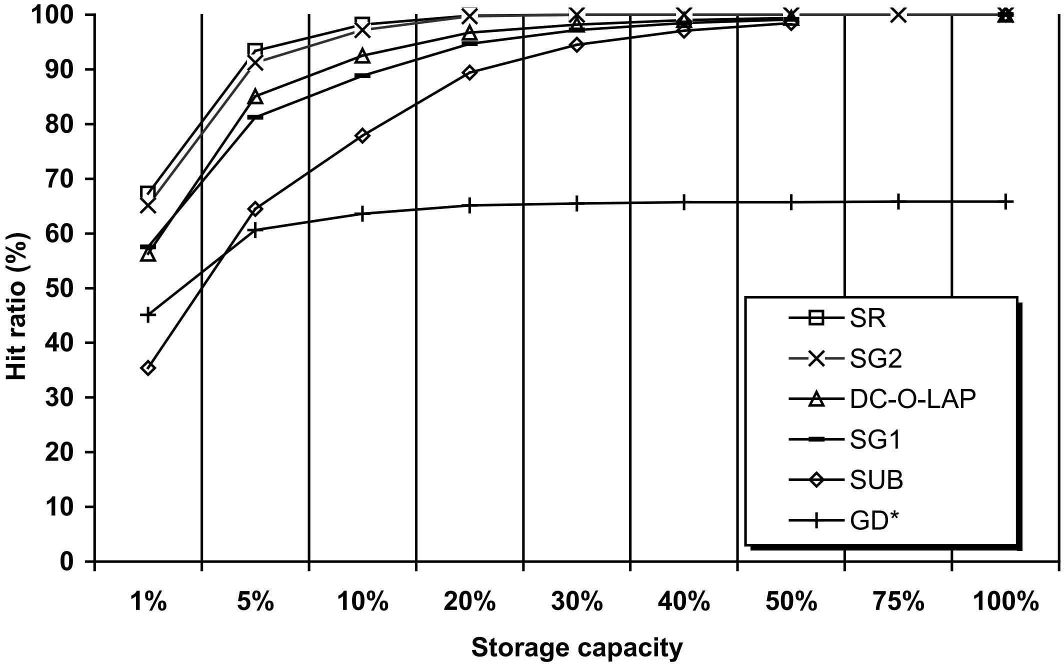


Figure 6.7 Hit ratios of all the methods for NEWS (*SQ* = 1, cache size from 1% to 100%)

Figure 6.8 compares the hit ratios of the major algorithms discussed in this paper for the NEWS trace under three cache capacity settings 1%, 5% and 10%.

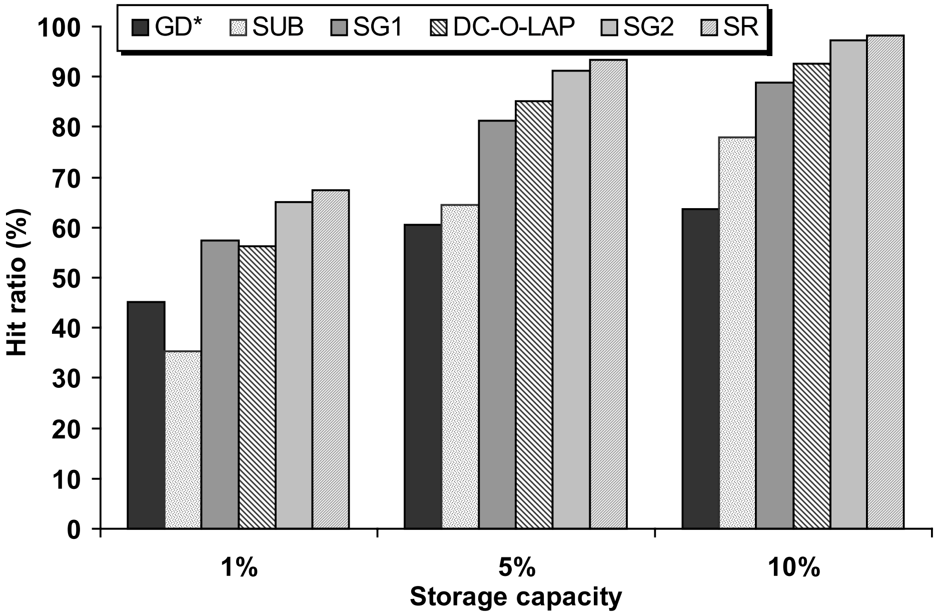


Figure 6.8 Hit ratios of all the methods for NEWS (*SQ* = 1, cache size 1%, 5% and 10%)

The only case in which any of our new approaches that incorporate subscription-based publish-time placement are worse than the access-based caching-only scheme GD* is when the cache capacity is low (1%) and GD* is compared to the simple subscription-based pushing-only scheme SUB. When the cache capacity is high, even SUB yields relatively higher hit ratio than GD*: 6% higher when the cache capacity is 5% and 22% higher when the cache capacity is 10%. Not trivially, the news trace is featured in that many user requests concentrate on few pages, which is biased to basic caching approaches in comparison to a more general trace (compare to Figures 6.15 and 6.16).

While the hit ratio increases with the capacity setting for any method, the relative performance ranks of the approaches are quite stable under different capacity settings. All the other subscription-based approaches shown in the figure outperform SUB under any setting.

SG2 and Subscription-Request (SR), which use the estimation of the number of requests of a page in the future, provide the highest hit ratios. The temporal analysis on accesses in SG2 does **not** provide extra benefit over SR, which illustrates the difficulty of combining the access recency in past and the frequency information of future usage. SG1 also combines subscription and access information for both publish-time and access-time placement, as do SG2 and SR. However, SG1 has a lower hit ratio than SG2 and SR because SG1 ignores the correlation between the subscriptions and the accesses of a page. The comparison between the performances of SG1, SG2 and SR demonstrates the importance of making use of the correlation between subscription and access when the correlation is strong.

DC-O-LAP has a hit ratio similar to SG1. When the storage capacity is high (5% or 10%), DC-O-LAP yields around 4% higher hit ratio than SG1. The superiority of SG2 and SR to DC-O-LAP shows that it is not necessary to apply a dual-caches method that uses the subscription and access information separately if users' accesses match their pre-defined interest perfectly well (e.g. SQ = 1). We examine other values of subscription quality (SQ) next.

To study the performances of our approaches under a much lower first time miss ratio, for example, assuming our algorithms are applied at reverse proxy servers, we generate another workload with first time miss ratio as low as 4.8%. The benefits by using our subscription-enhanced content delivery approaches are also pronounced, as demonstrated in appendix B.

6.1.4.2 Influence of Subscription Quality

Focusing on the 5% capacity level, Figure 6.9 reveals the effect of subscription quality (SQ) that is the probability for a user to read a page in the notification list. All the approaches are affected by SQ, except for GD* which does not use the subscription information at all.

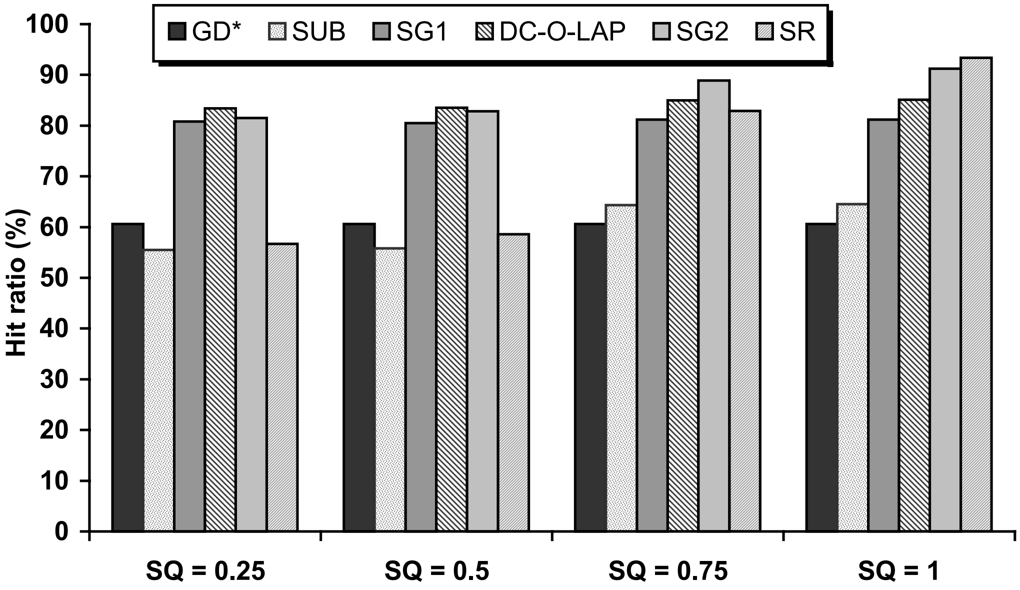


Figure 6.9 Hit ratios of the algorithms with different subscription qualities (storage capacity = 5%)

SR, the best approach when the subscription information is perfectly accurate (SQ = 1), is most affected by SQ, and its superiority disappears quickly as SQ decreases. SR and SUB are worse than GD* if SQ ≤ 0.5. By incorporating the analysis of access recency information, SG2 remains highly effective when SQ varies. When SQ ≤ 0.5, whether one is using the sum (SG1) or the difference (SG2) of the number of subscriptions and that of accesses becomes less important because the number of subscriptions dominates the frequency factor in Equation 4.1.

Both SG1 and DC-O-LAP are not sensitive to SQ. As SQ decreases, DC-O-LAP, SG1 and SG2 algorithms have similarly good performances, although DC-O-LAP has about a 2% higher hit ratio over SG2 when SQ is as low as 0.25. The power of DC-O-LAP, which takes into consideration the mismatch between subscription and access patterns is partially demonstrated through this result, even though the degree of mismatch is not very high because all the accesses are assumed to be driven by notifications (F = 1).

6.1.4.3 Hit Ratio Versus Time

Figure 6.10 demonstrates the average hit ratio of three algorithms in every hour, given that the subscription quality is 1 and the cache capacity setting is 5%. SG2, the best approach that combines publish-time and access-time placement in general, is compared against the simple subscription-based pushing-only method SUB and the access-based caching-only method GD*.

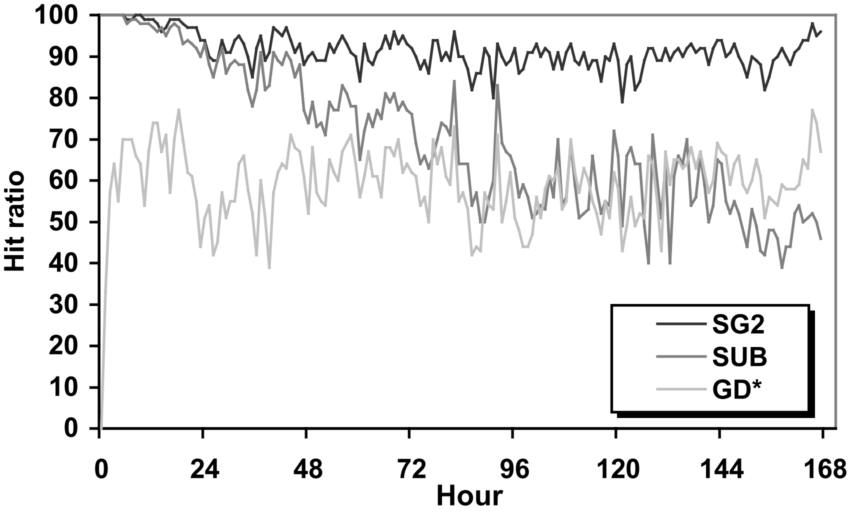


Figure 6.10 Average hit ratio *H* hourly (SQ = 1, capacity = 5%)

After the first couple of hours, GD* behaves stably. At the beginning, SUB has a high hit ratio by proactively delivering contents to the content servers at publish time before users request them. However, the hit ratio of SUB drops with time because SUB only uses static subscription information to deliver contents at publish time but totally ignores the opportunity to deliver contents at access time. In addition, SUB does not adjust its evaluation of pages according to actual request patterns. SG2 keeps a high hit ratio by combining the analysis of the access pattern and the subscriptions in the replacement algorithm.

6.1.4.4 Workload at the Publishing Site

Figure 6.11 demonstrates the workload in terms of the number of requests at the publishing site.

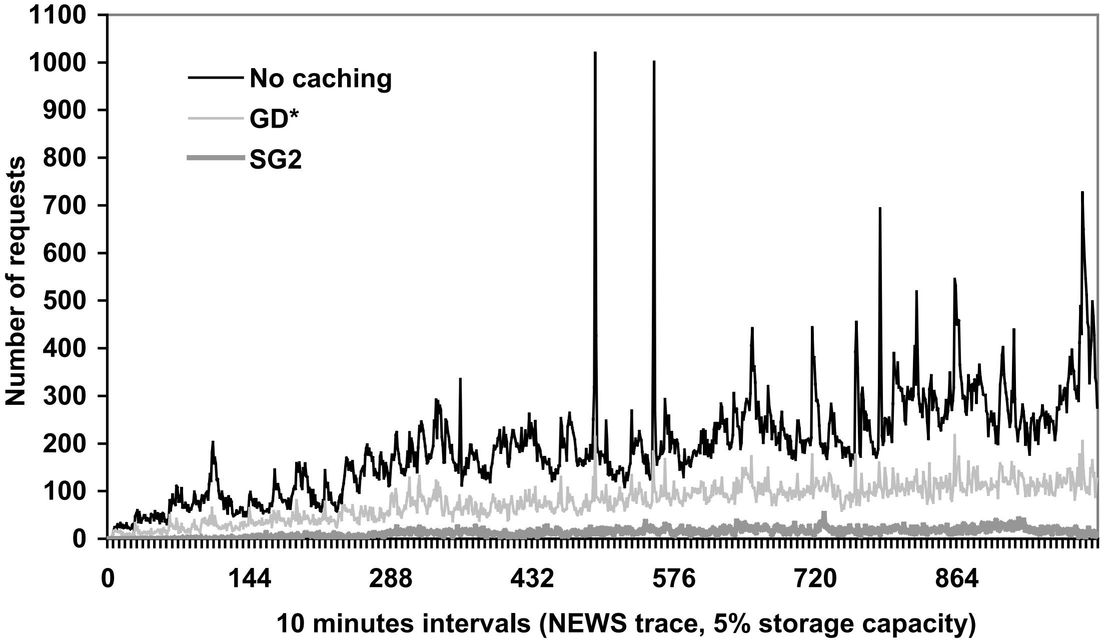


Figure 6.11 Workload at the publishing site

When not using any content delivery techniques so that every user request has to go to the publishing site, the publishing site needs to serve about 200 or 300 requests in total from 100 sites in every 10 minutes. More importantly, flash crowds, which are the burst in requests, exist in the original user requests. The caching algorithm GD* is useful to reduce the workload at the publishing site and remove the flash crowds. However, SG2 is much more successful than GD* for shifting the workload from the publisher to multiple distributed content servers.

6.1.4.5 Traffic Overhead for NEWS

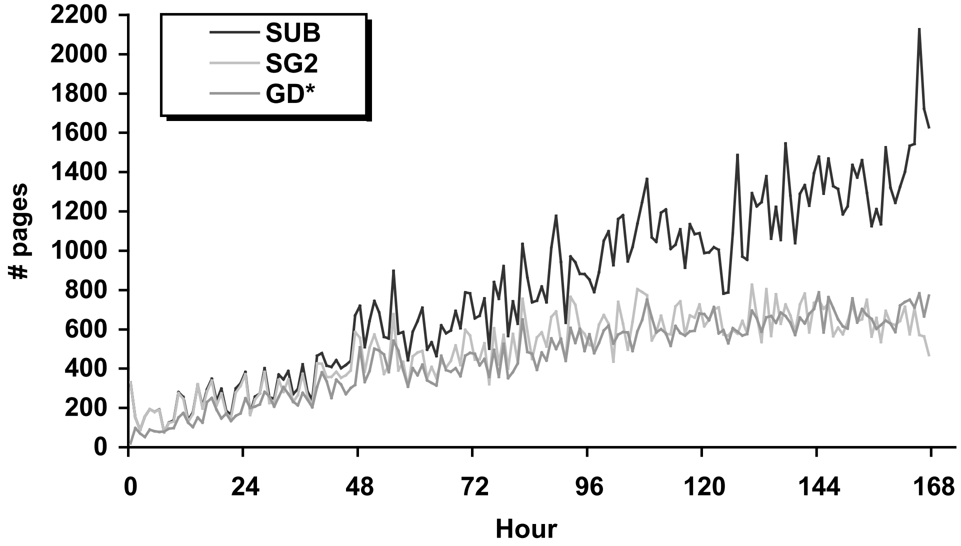
The publish-time module can use either of the following two schemes to push contents:

(1) *Always Pushing*: the publish-time module transfers all the relevant matched pages to a server when pages generated are matched to the subscriptions from the server; the server then decides whether to store each page in its local cache based on the replacement algorithm. Bandwidth is wasted if the server decides not to store a page.

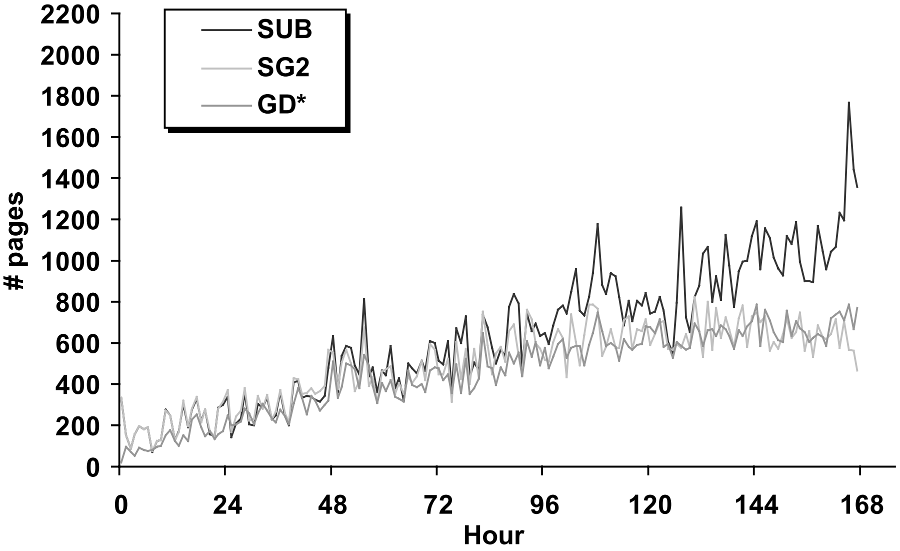
(2) *Pushing When Necessary*: the publish-time module notifies the server of the meta-information such as page size when a page matches the subscriptions from the server; then the server performs its evaluation to decide whether to store the page and sends the result to the publish-time placement module; if the reply is “will store it in cache”, the publish-time placement module notifies the publisher to forward the page to the server. This scheme is designed to reduce unnecessary pushing of pages from the publisher to the server, but it can mean additional control messages. The cost of control messages is ignored in the following analysis.

The total amount of traffic within the whole network at any time is the sum of two parts. One part of traffic comes from publish-time placement, and another part is for fetching contents on cache misses. The caching approach GD* does not consume bandwidth for publish-time placement, while the other two approaches do. However, the advantage of the subscription-based approaches in reducing cache misses implies lower traffic overhead due to cache misses as compared to GD*.

Figures 6.12 and 6.13 show the total traffic for pushing pages and fetching pages on cache misses versus time, in terms of the number of pages or the total number of bytes, when using GD*, SUB and SG2, considering each one of the above two pushing schemes.

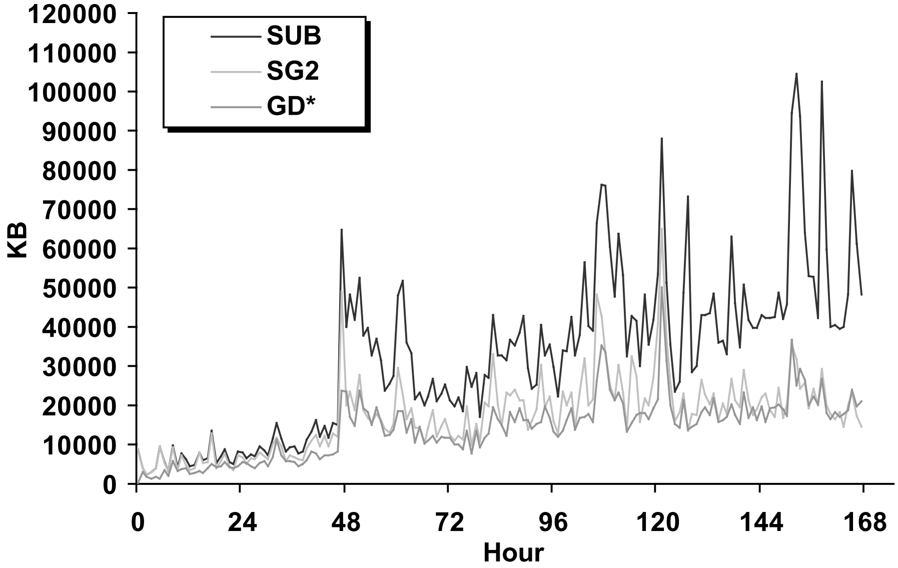


(a) ***Always-Pushing*** **scheme**

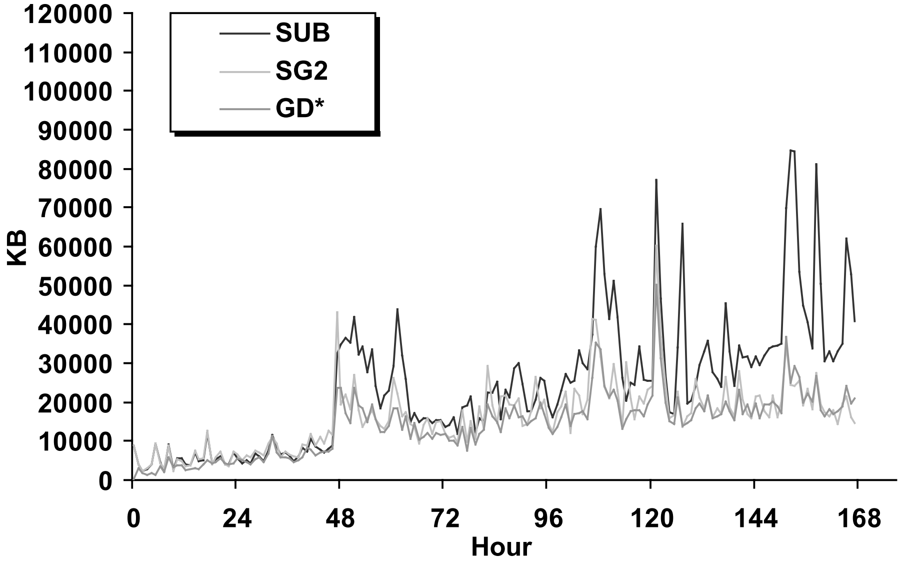


(b) ***Pushing-When-Necessary*** **scheme**

Figure 6.12 Traffic in number of pages (SQ = 1, capacity = 5%)



(a) ***Always-Pushing*** **scheme**



(b) ***Pushing-When-Necessary*** **scheme**

Figure 6.13 Traffic in bytes (SQ = 1, capacity = 5%)

The traffic overhead of GD* does not change with pushing scheme so it can be used as the baseline to compare the two pushing schemes. Interestingly, SG2 is not sensitive to the two options, Always-Pushing and Pushing-When-Necessary. The insensitivity of SG2 implies there is a high probability for SG2 to store a new page. Using any option, the traffic overhead of SG2 is comparable to the access-time pull scheme GD*.

Using any pushing scheme, SUB always introduces the highest traffic overhead among the three algorithms. This is because that SUB suffers from fetching-on-miss due to its low hit ratio as GD* does, and it consumes bandwidth in publish-time pushing as SG2 does. When using the Pushing-When-Necessary scheme, the difference between the curves of SUB and GD* is smaller than using Always-Pushing, which means the former benefits SUB a lot in reducing the traffic overhead in publish-time pushing. The above observations hold in terms of both traffic metrics, pages and kB.

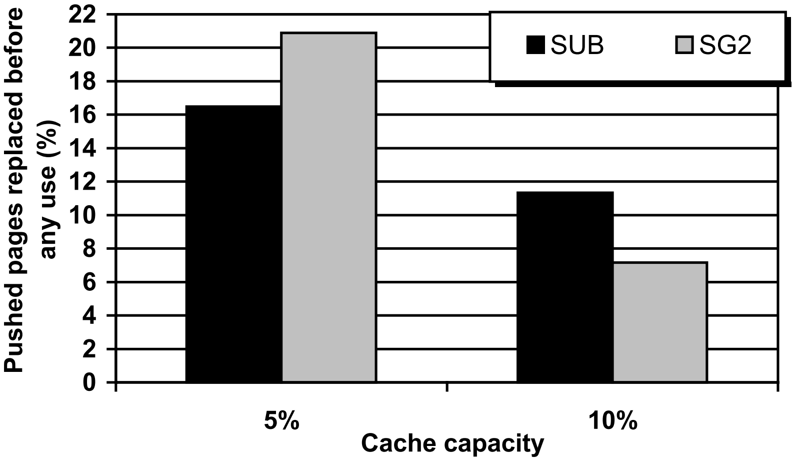


Figure 6.14 Fraction of pushed pages that are evicted before the first request for NEWS

Figure 6.14 shows the fraction of the pages that are stored in local servers at publish time but evicted during replacements before the pages are requested by any user. We call these pages as “wrongly placed”. At the 5% cache capacity setting, the fraction of “wrongly placed” pages is not high when using SUB. Therefore the higher network traffic of SUB than that of SG2 in Figures 6.12 and 6.13 is mainly resulted from the network cost for fetching on cache misses.

Overall, SG2 performs very well in terms of improving hit ratio as well as keeping traffic low, and being relatively insensitive to several parameters.

6.1.5 Performance of Major Algorithms for ALTERNATIVE Trace

The experimental results in this section demonstrate the performances of our approaches for a web reference stream with a more uniform popularity distribution than NEWS. Namely, the requests distribute more uniformly among pages in ALTERNATIVE than in NEWS.

6.1.5.1 Comparison of the Approaches Under Different Cache Sizes

Figure 6.15 and Figure 6.16 show the hit ratios of different methods for trace ALTERNATIVE. For both NEWS and ALTERNATIVE, the common observation is that our new approaches provide a much higher hit ratio than the access-based caching-only scheme GD*. Even the only bad case for NEWS (SUB for 1% cache capacity) does not exist for ALTERNATIVE. There are certain changes in the results for ALTERNATIVE from those for NEWS.

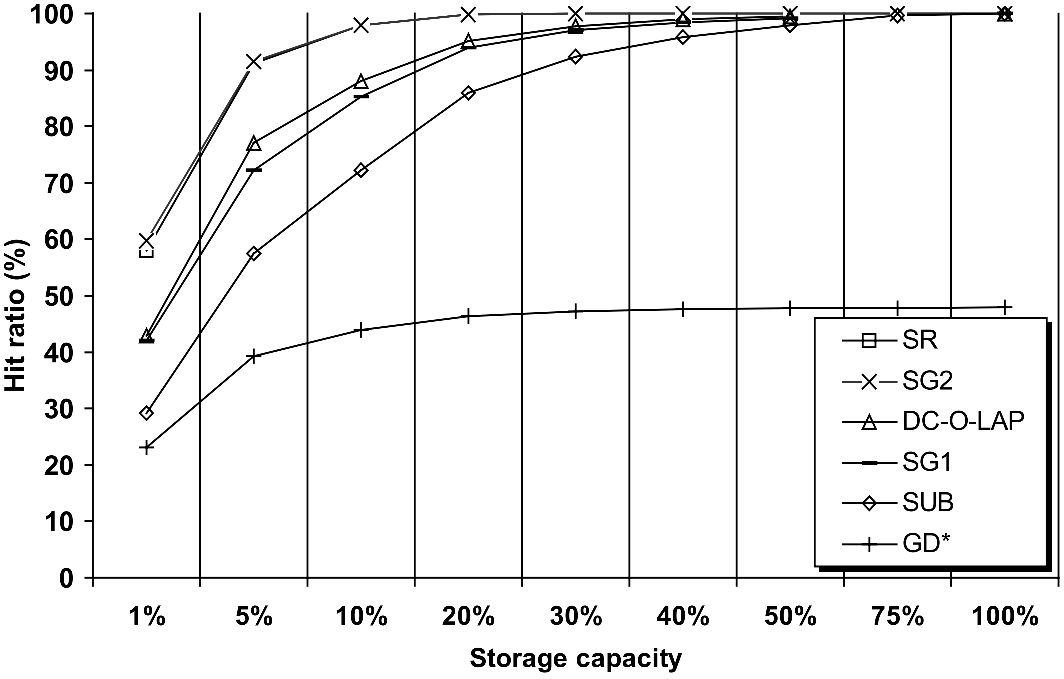


Figure 6.15 Hit ratios of all the methods for ALTERNATIVE (*SQ* = 1, capacity from 1% to 100%)

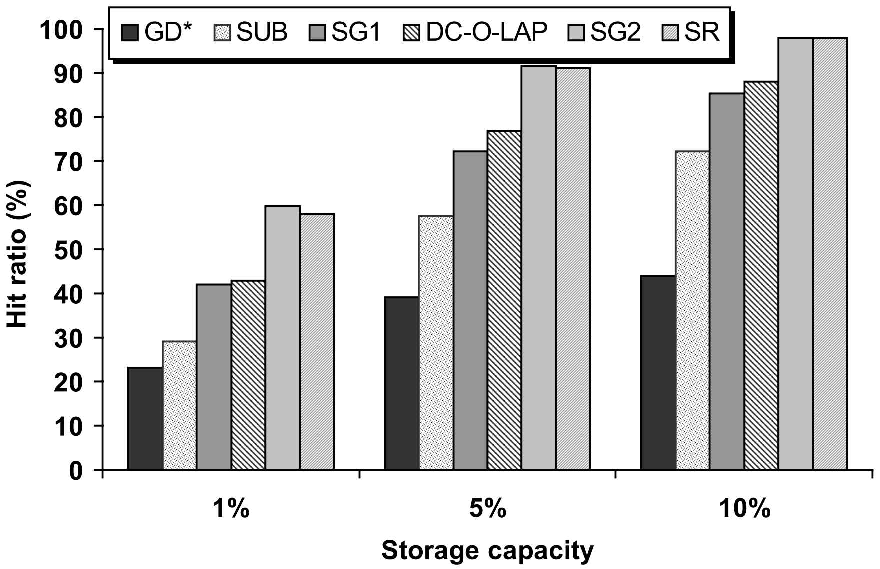


Figure 6.16 Hit ratios of the approaches for ALTERNATIVE (SQ = 1, cache size 1%, 5% and 10%)

When α becomes smaller (α = 1.0, in ALTERNATIVE), the hit ratio of the caching approach GD* is much lower than that when α is 1.5 (in NEWS), as with all other content distribution schemes. The degradation in hit ratio is not surprising since a more uniform popularity distribution implies fewer repeated reference to the same page, and the first time miss ratio at a server in ALTERNATIVE is higher (around 50%). However, the relative improvements using subscription-based pushing-enhanced methods are much higher when α is 1.0 than when α is 1.5, as summarized in Table 6.2.

Table 6.2 Relative improvements in hit ratio of different approaches over GD* (%) (capacity = 5%)

Method α	SUB	SG1	SG2	SR	DC-LAP
1.5	6	34	50	54	40
1	47	84	133	133	96

The much higher relative improvements of our new methods for ALTERNATIVE mean that non-homogeneous request streams (characterized by low α) make the publish-time placement module more important. The higher importance of pushing for ALTERNATIVE is also revealed in the substantial improvement in performance of the pushing-only scheme SUB for ALTERNATIVE versus NEWS.

6.1.5.2 Influence of Subscription Quality

As shown in Figure 6.17, the ranks of different approaches as a function of subscription quality are similar for ALTERNATIVE and NEWS. One major distinction from the results for NEWS is that the hit ratio of SG2 drops more quickly and it is even worse than SG1 when SQ ≤ 0.5. One possible reason is that since the access frequencies of pages are getting similar with smaller α, the subscription frequency dominates the frequency factor in the evaluation method in SG2. Therefore, the accuracy of subscription information becomes more important for SG2.

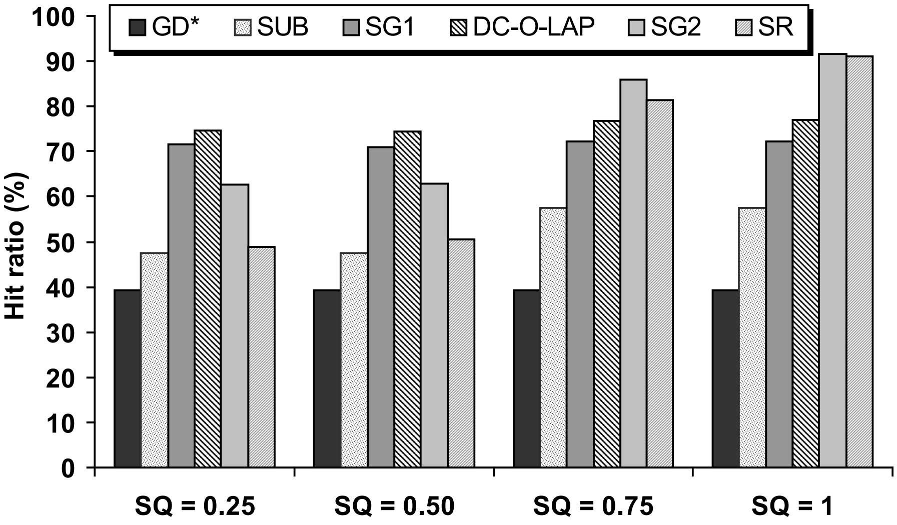


Figure 6.17 Hit ratios with different SQ for ALTERNATIVE trace (capacity = 5%)

6.1.5.3 Hit Ratio Versus Time

Figure 6.18 demonstrates the hit ratios of GD*, SG2 and SUB over time. The observations about the temporal performances of the three algorithms for trace ALTERNATIVE are similar to those for NEWS. However, the hit ratio of SUB is always above GD* for ALTERNATIVE, even though it is still downward with time.

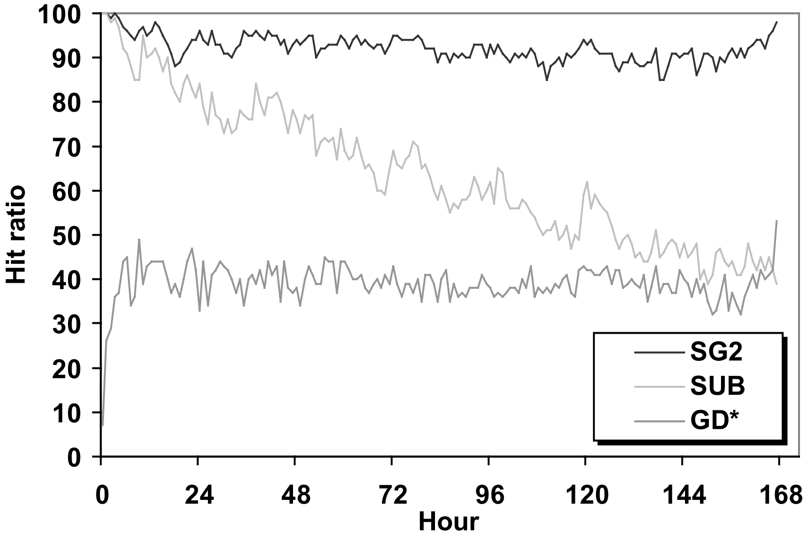
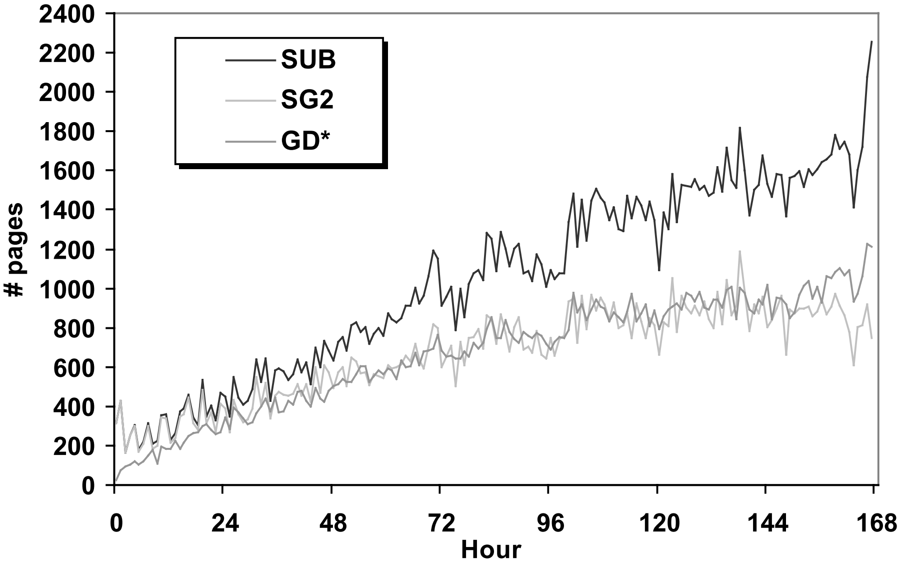


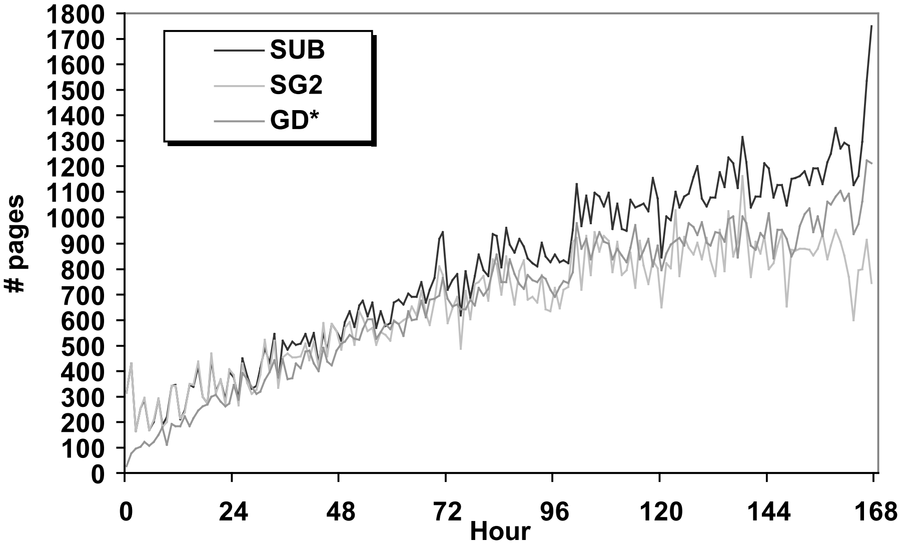
Figure 6.18 Average *H* hourly (SQ = 1, capacity = 5%)

6.1.5.4 Traffic Overhead for ALTERNATIVE

Figures 6.19 and 6.20 demonstrate the traffic overheads of GD*, SUB and SG2.

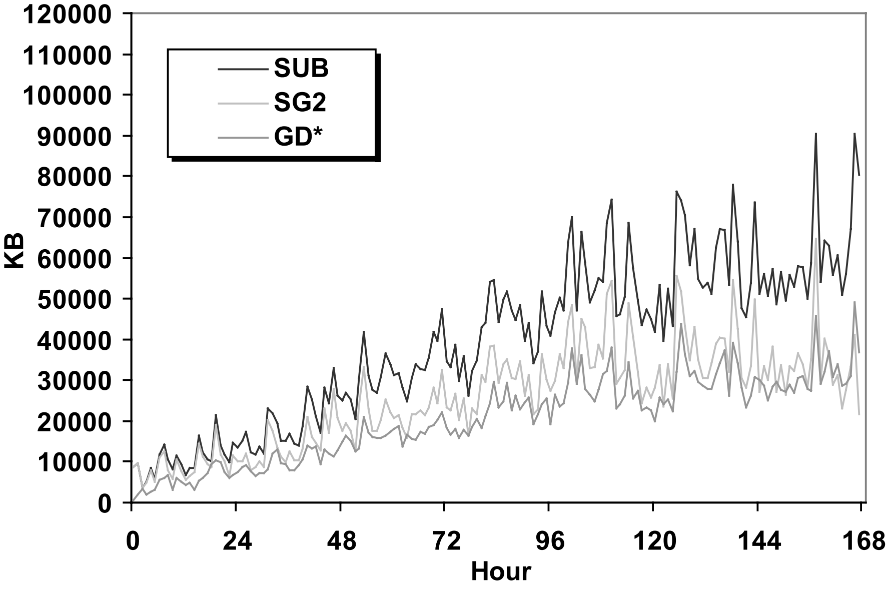


(a) *Always-Pushing* scheme

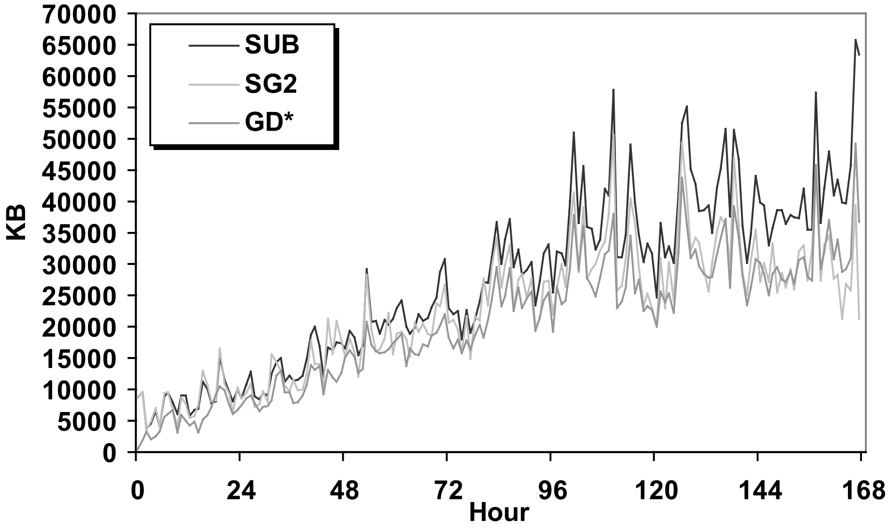


(b) *Pushing-When-Necessary* scheme

Figure 6.19 Traffic in number of pages for ALTERNATIVE (SQ = 1, capacity = 5%)



(a) *Always-Pushing* scheme



(b) ***Pushing-When-Necessary*** **scheme**

Figure 6.20 Traffic in kilobytes for ALTERNATIVE (SQ = 1, capacity = 5%)

Compared to the results for NEWS, the difference between the network overhead of SG2 and that of GD* is smaller. The overhead of SG2 in terms of number of pages is even lower than GD* at the later stage of the simulation. The relative difference between SUB and GD* is also lower than that for NEWS. The change is due to the higher relative hit ratio using SUB and SG2 over GD*.

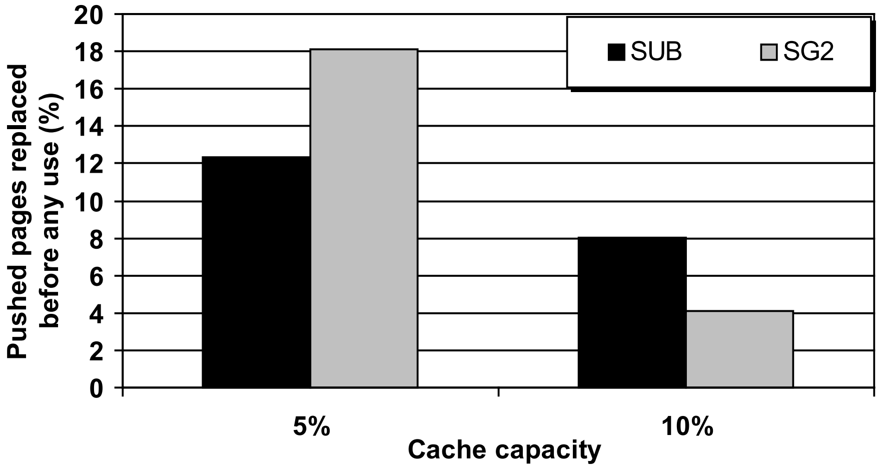


Figure 6.21 Fraction of pushed pages that are evicted before the first request for ALTERNATIVE

Figure 6.21 shows that as for NEWS trace (Figure 6.14), at the 5% cache capacity, the fraction of "wrongly placed" pages is not high when using SUB. Therefore the higher network traffic of SUB than that of SG2 is resulted from the network cost for fetching in cache misses.

6.2 Comparing Algorithms for More General Scenarios

The experiments in this section measure the performances of six major algorithms under a more general scenario in which not only not every notification drives an access, but not all requests are driven by notifications. GD* is the baseline algorithm. Subscription-GD*-1 (SG1) is proposed for pure publish-subscribe services and evaluated in the new scenario. Restrictive-Subscription-GD*-2 (RSG2) is the adapted version of SG2 to be suited in the new scenario. The three approaches Dual-Caches Partitioned by Operation with Limited Adaptive Partition (DC-O-LAP), Hybrid-User-GD* (HUG) and Dual-Caches Partitioned by Type of Access (DC-TA) are presented for the more general scenarios in which some user accesses are not driven by notifications and/or subscribers do not request all contents in the notification messages.

To investigate the importance of subscriptions in content delivery systems under a given user model (quantified by {F, SQ} as defined in section 5.2.3), the performance of any of our subscription-based approaches is expressed as the relative improvement in hit ratio *H* as compared to the baseline caching-only approach GD*. The relative improvement *I* is defined in Equation 6.2. A positive value of *I* indicates an improvement of a method over GD*.

$$I_i = \frac{H_{GD*} - H_i}{H_{GD*}} \cdot 100\% \qquad \text{- Equation 6.2}$$

Where

I_i : the relative improvement of method i

H_i : global hit ratio of method i

H_{GD*} : global hit ratio of GD *

In this study, the cache size is set to be 5% of the total number of unique bytes requested at the server in the whole simulation. The performances of the approaches are tested under different user models that are characterized by the two global parameters, F and SQ.

Our experiments focus on three comparisons. The first comparison is between RSG2 and HUG, the two approaches that combine subscription and access information into a single evaluation function. The second comparison is between two Dual-Caches approaches that divide and manage a server's cache in different ways. Finally, we examine the best approach under different user models in terms of F and SQ.

6.2.1 Comparing RSG2 and HUG

The experiments in section 6.1 identified SG2 as one of the best content delivery and caching approaches when F is 1. RSG2 and HUG, the two variants of SG2, differ in whether the two types of accesses, notification-driven accesses and general browsing, are distinguished in the access frequency analysis. Therefore, the two approaches are the same when F is 1, in which case all accesses are notification-driven.

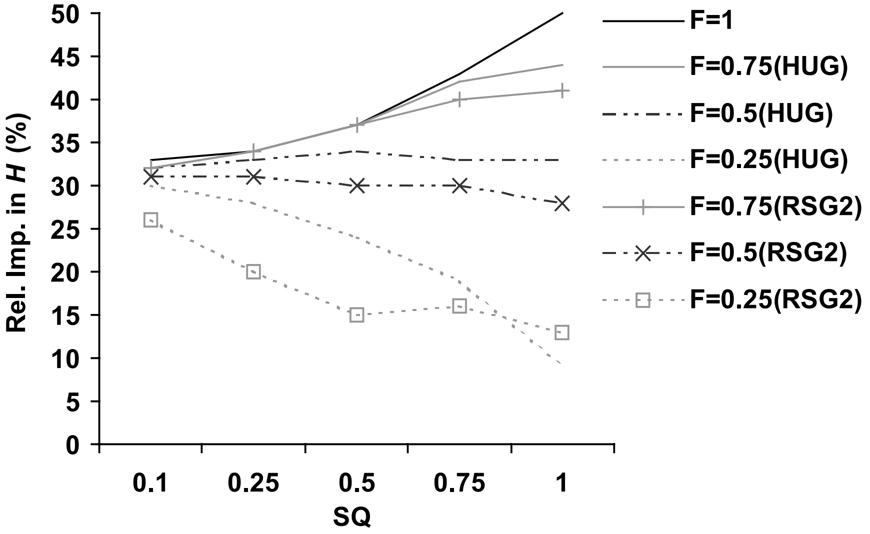


Figure 6.22 Performances of RSG2 and HUG

As shown in Figure 6.22, RSG2 and HUG improve the hit ratio over GD* in all tested cases. The superiority of HUG to RSG2 in most cases demonstrates the importance of distinguishing access type in evaluating pages.

A larger *F* indicates that most accesses are driven by notifications and thus implies greater importance of the subscription analysis. The positive correlation between *F* and the performances of RSG2 and HUG in Figure 6.22 exhibits that both approaches are able to use the subscription information adequately to enhance content delivery.

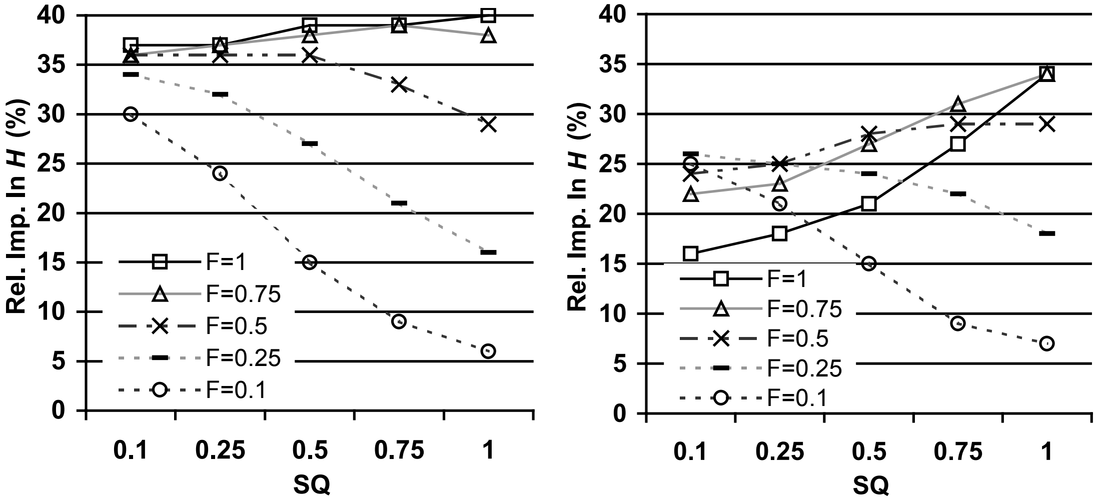
Interestingly, higher subscription quality itself does not always benefits the performances of the approaches. When F is 0.25, the performances of RSG2 and HUG degrade with SQ in general. A combination of small F and large SQ is likely to result in a small number of subscriptions using Equation 5.2, which reduces the importance of subscription information.

6.2.2 Comparing Two Dual-Caches Algorithms: DC-O-LAP and DC-TA

The experiments in this section compare two Dual-Caches algorithms, Dual-Caches Partitioned by Operation with Limited Adaptive Partition (DC-O-LAP) and Dual-Caches Partitioned by Type of Access (DC-TA). The first experiment is conducted between DC-O-LAP (adaptive partition between 25% and 75%) and DC-TA with 50%-50% partition between two cache portions. Then we study the influence of cache partition on the performance of DC-TA and examine the optimum partition for DC-TA under different F.

6.2.2.1 Comparing DC-O-LAP and DC-TA

Figure 6.23 shows the performances of the two dual-caches approaches. Both DC-O-LAP and DC-TA yield better results in terms of hit ratio than GD* in all cases. As for RSG2 and HUG, the performances of DC-O-LAP and DC-TA become worse with SQ when F is 0.25. As explained before, this is because a combination of small F and large SQ results in little subscription information available in the workload.



(a) Performances of DC-O-LAP **(b) Performance of DC-TA**

Figure 6.23 Comparison of performances of two Dual-Caches approaches

Under almost all settings of F and SQ, DC-O-LAP yields higher gains than DC-TA. Recall that the major distinction between DC-O-LAP and DC-TA is the criterion to partition pages into two cache portions. DC-O-LAP groups all pages that have not been referenced in one portion, and other pages in another portion. In contrast, when a placement is triggered on a cache miss, DC-TA may put the new page into either NP or BP (see section 4.4.5) according to the access type. Consequently, DC-TA may evaluate a page using different rules at different reference times of the page. We conjecture that the partition rule based on type of accesses may be one reason for the worse performance of DC-TA.

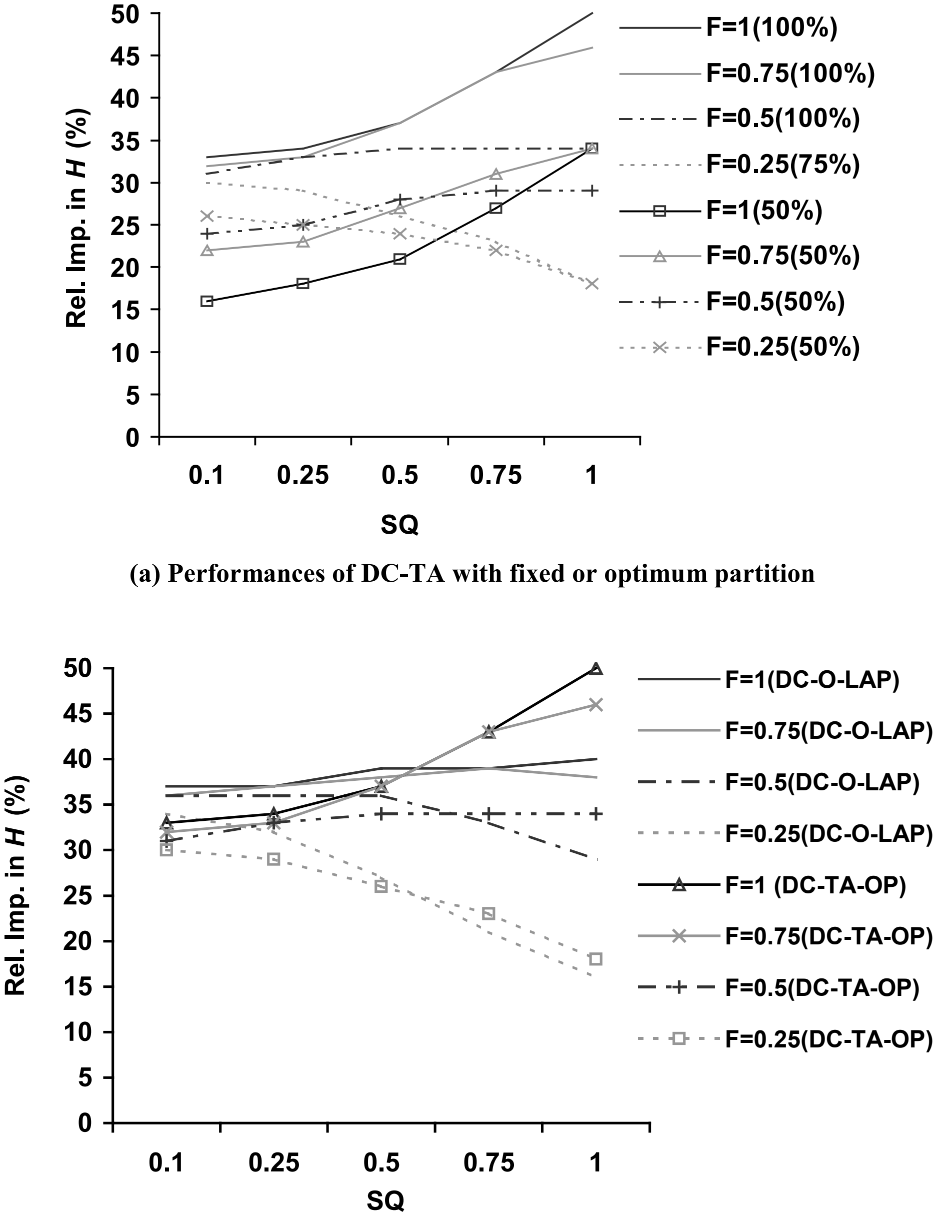
6.2.2.2 Influence of Cache Partition on DC-TA

Different from RSG2, HUG, and DC-O-LAP, the performance of DC-TA does not increase with F. Given an SQ, DC-TA achieves the best or the second best result when F is 0.5. Since DC-TA uses a 50%-50% partition on a server's cache, we conjecture that the good performances of DC-TA when F is 0.5 are resulted from the suitable cache partition under this F value. However, a 50%-50% partition may not be good for traces with other F values. Recall that DC-O-LAP adaptively updates its partition based on the publishing pattern and the request pattern over time. The intelligent storage partition may be another reason for the superiority of DC-O-LAP to DC-TA in Figure 6.23.

To study the influence of the cache partition, DC-TA is tested under different partitions for each F. Based on our experimental results, the optimum storage fraction assigned to notification-based portion are 100%, 100%, 100%, and 75% for F of 1, 0.75, 0.5 and 0.25 respectively. In particular, when at least half of the accesses are driven by notifications ($F \geq 0.5$), it is better to use the whole cache for the publish-time placement and the access-time placement based on subscriptions and notification-driven accesses only.

The major benefit of notification cache portion comes from proactive placement at publish-time in reducing cold misses, while the benefit of browsing cache portion is from serving repeated requests in general browsing. Therefore, the optimum partitions for DC-TA indicate the higher importance of reducing cold misses than serving repeated general browsing when the fraction of notification-driven accesses is not lower than 0.5. The optimum fraction value of the notification-driven portion, denoted as OF, is positively correlated to F and is always larger than F for $F < 1$, which means that giving higher importance to the notification-driven accesses and subscriptions benefits all users in reducing response time.

Figure 6.24(a) exhibits a dramatic improvement using DC-TA with Optimum Partition or DC-TA-OP over DC-TA with fixed 50%-50% partition. Figure 6.24(b) compares DC-O-LAP and DC-TA-OP. Using the optimum partitions, DC-TA-OP has comparable performances to DC-O-LAP in many cases. When SQ is high, DC-TA-OP yields the same or even better results than DC-O-LAP. The results in Figure 6.24 support the hypothesis that adaptive partition based on access patterns is important to dual-caches approaches.



(a) Performances of DC-TA with fixed or optimum partition

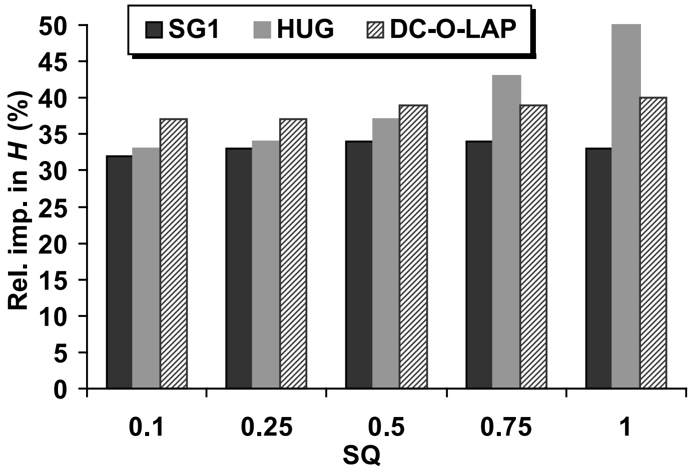
(b) Performances of DC-O-LAP and DC-TA-OP

Figure 6.24 Influence of cache partition to DC-TA

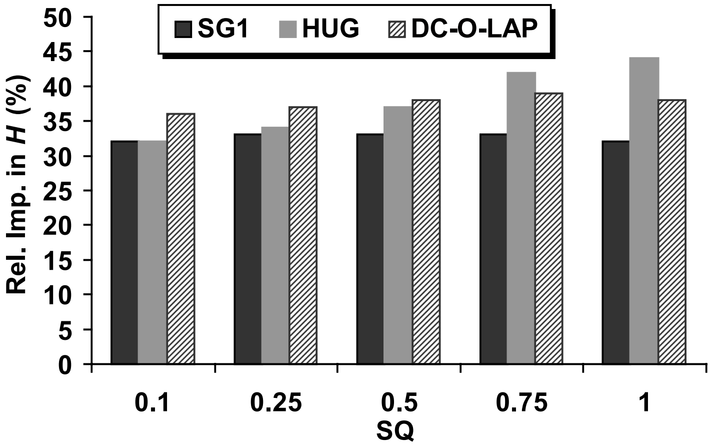
Although the performance of DC-TA with the optimum partition is similar to that of DC-O-LAP, the performance of DC-TA is highly sensitive to the cache partition. In practice, it is hard to know the user model and set a suitable partition in advance, therefore DC-O-LAP is a better model than DC-TA in general.

6.2.3 Comparing SG1, HUG and DC-O-LAP

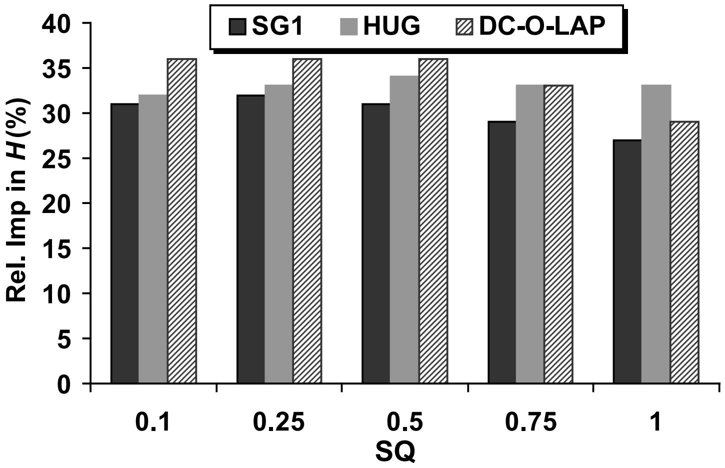
The experiments in sections 6.2.1 and 6.2.2 demonstrate that HUG outperforms RSG2, and DC-O-LAP outperforms DC-TA under most settings of F and SQ. The following figures compare the performances of HUG and DC-O-LAP to that of SG1, one of our approaches for pure notification-driven accesses.



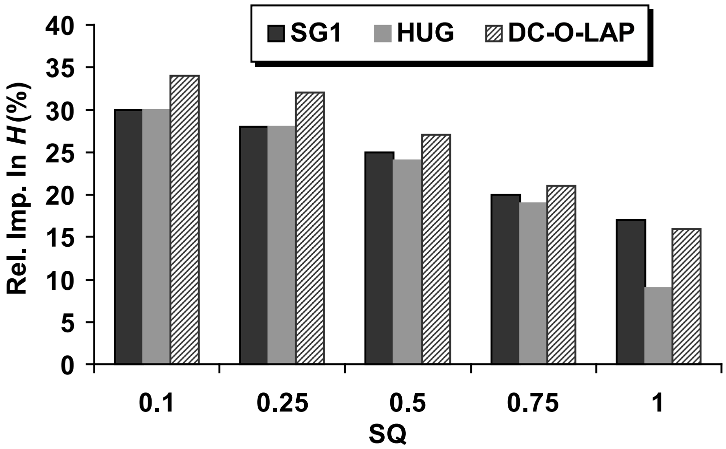
(a) F = 1



(b) F = 0.75



(c) F = 0.5



(d) F = 0.25

Figure 6.25 Performances of SG1, HUG and DC-O-LAP

As shown in Figure 6.25, all of the three approaches improve the hit ratio over GD* under all settings of F and SQ. Table 6.3 summarizes the winners for different values F and SQ.

Table 6.3 Winners for the NEWS trace at different F and SQ

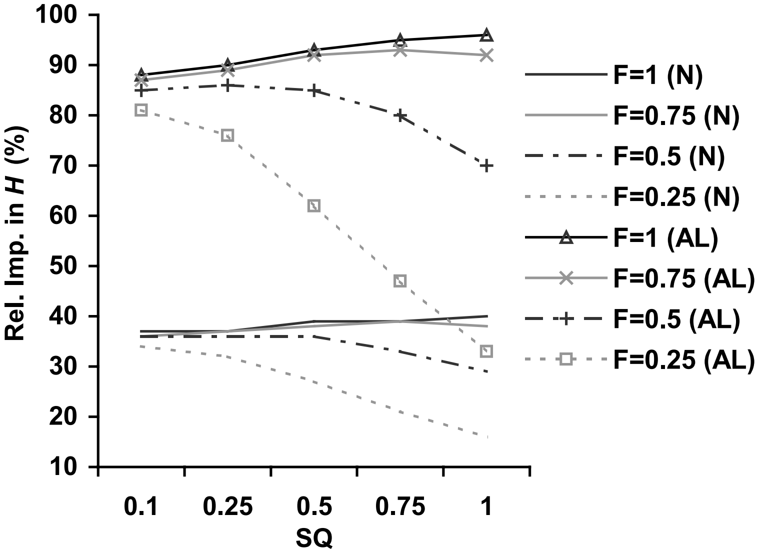
SQ F	0.1	0.25	0.5	0.75	1
0.25	DC-O-LAP	DC-O-LAP	DC-O-LAP	DC-O-LAP	SG1
0.5	DC-O-LAP	DC-O-LAP	DC-O-LAP	HUG/DC-O-LAP	HUG
0.75	DC-O-LAP	DC-O-LAP	DC-O-LAP	HUG	HUG
1	DC-O-LAP	DC-O-LAP	DC-O-LAP	HUG	HUG

When both F and SQ are high (F ≥ 0.5, SQ > 0.5), HUG yields the highest relative gains over GD* among the three approaches, hence HUG is best when most requests follow the users' stated interest. DC-O-LAP is the winner for most settings of F and SQ when either F or SQ is low. As the simplest approach among the three, SG1 yields comparably good results only when F is low and SQ is high.

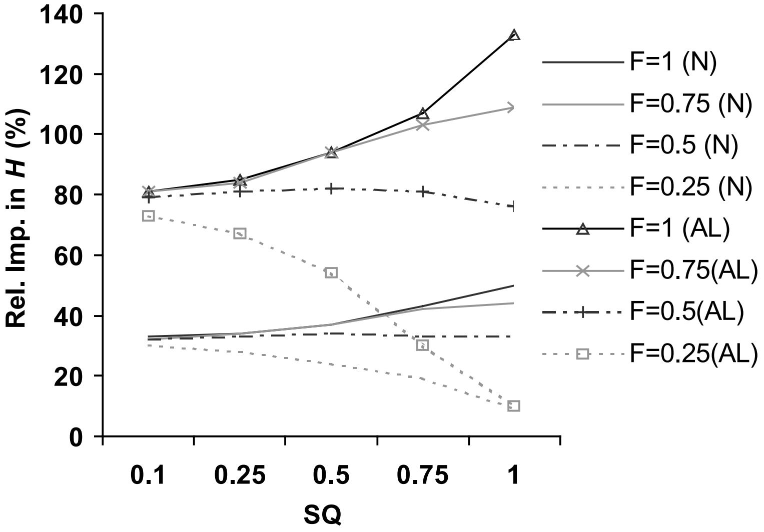
6.2.4 Analyzing the Influence of Homogeneity Parameter α

The above experiments for the general user models are for trace NEWS (defined in chapter 5) that is featured in high concentration of requests on a few of very popular documents. This section studies the performances of the approaches using trace ALTERNATIVE (defined in chapter 5) that models a more general request pattern.

Figure 6.26 shows the performances of DC-O-LAP and HUG, the two best algorithms that are presented in this study, in the simulations using the two traces. Using ALTERNATIVE, the improvements in hit ratios of DC-O-LAP and HUG over that of the baseline algorithm are much more pronounced than using NEWS.



(a) Performances of DC-O-LAP for two traces (N is NEWS and AL is ALTERNATIVE)



(b) Performances of HUG for two traces (N is NEWS and AL is ALTERNATIVE)

Figure 6.26 Comparing performances of algorithms for two traces

The results for F = 1 were introduced in section 6.1.5. Figure 6.26 confirms that our algorithms yield greater superiority to GD* for the trace with more general popularity distribution of pages in the scenarios that not all requests are driven by notifications.

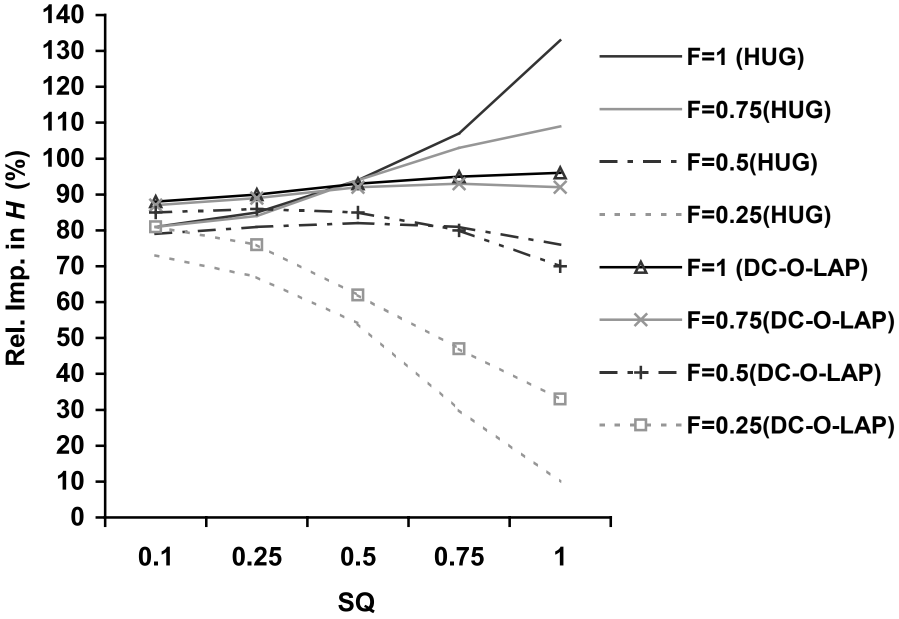


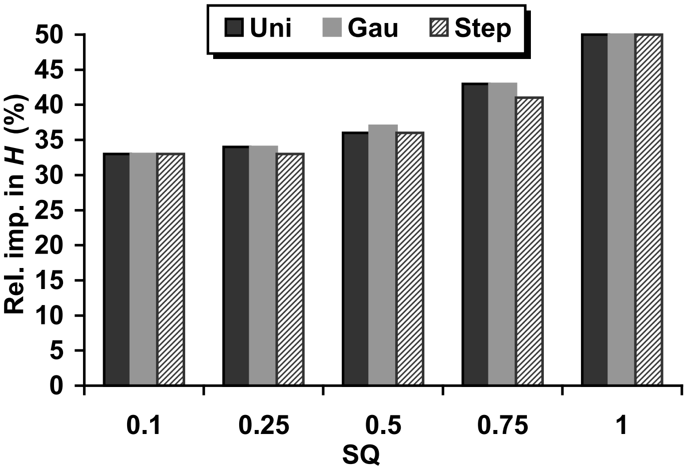
Figure 6.27 Comparing performances of DC-O-LAP and HUG for ALTERNATIVE

Figure 6.27 compares our two best approaches DC-O-LAP and HUG for the trace ALTERNATIVE. In consistence with the results for NEWS trace, HUG works better when both F and SQ are high; otherwise, DC-O-LAP is better than HUG.

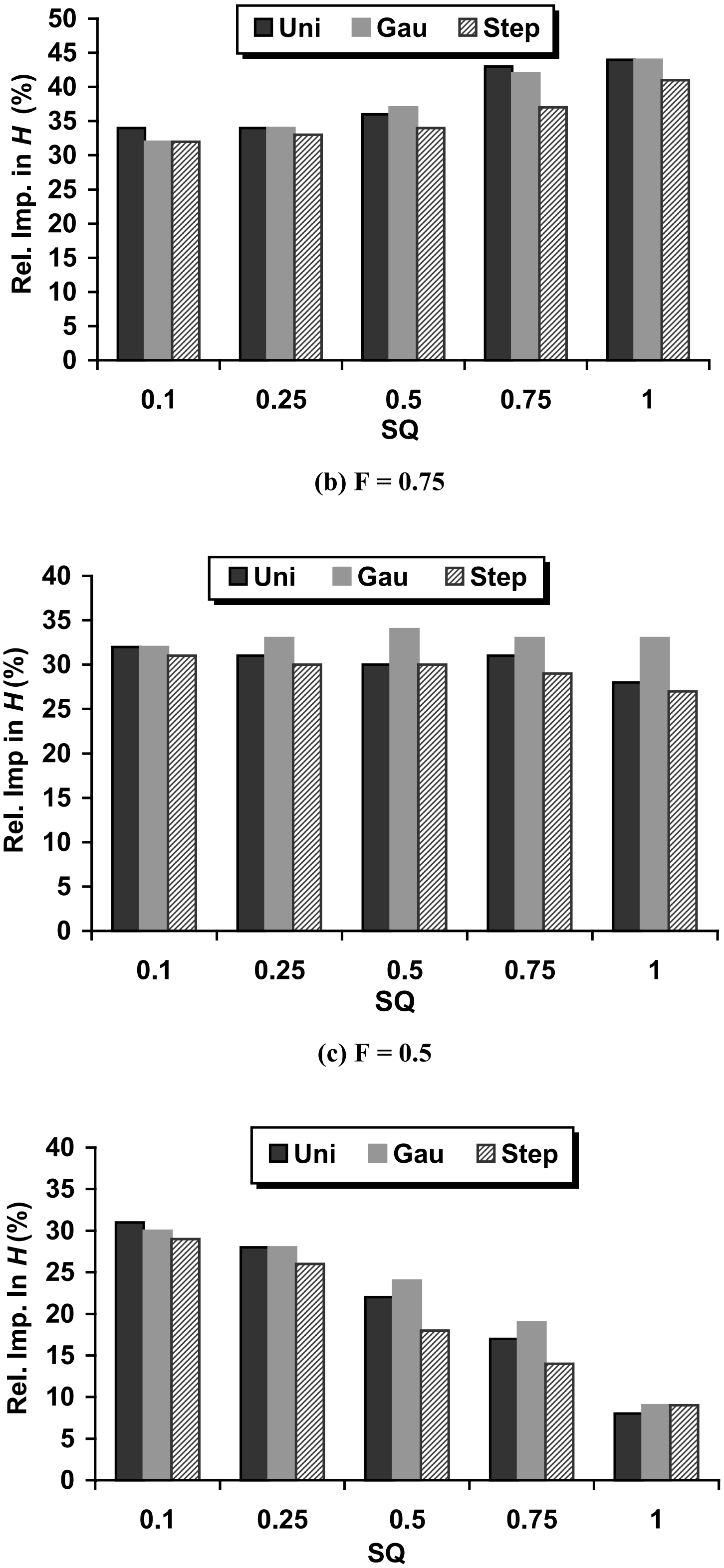
6.2.5 Analyzing the Influence of Distribution Functions for $\{F_{i,j}\}$ and $\{SQ_{i,j}\}$

The above experiments are for the workloads that are generated using Gaussian distribution to model $\{F_{i,j}\}$ and $\{SQ_{i,j}\}$ (defined in Equation 5.2). We also developed the workloads using two other distribution functions, uniform distribution and stepwise distribution (see Figure 5.10). The experimental results using the two functions are similar to those using Gaussian distribution, therefore we show only the performances of the two best algorithms HUG and DC-O-LAP for the workloads generated using the three distribution functions.

As shown in Figures 6.28 and 6.29, the results that use the uniform distribution to model $\{F_{i,j}\}$ and $\{SQ_{i,j}\}$ are very close to that using the Gaussian distribution. For the results using the step function, the hit ratios is lower than using the Gaussian function. When switching from the Gaussian function to the step function, HUG is affected more than DC-O-LAP.



(a) F = 1

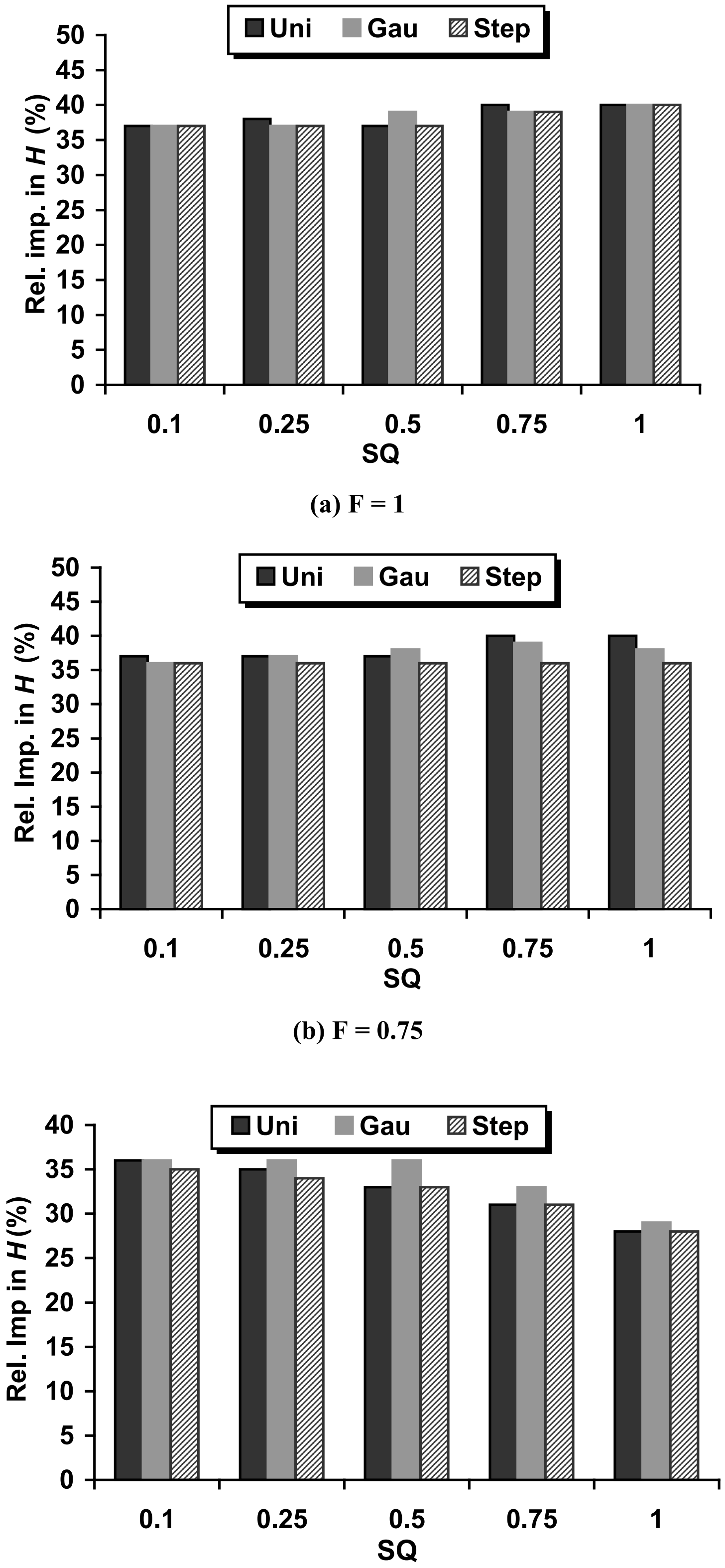


(b) F = 0.75

(c) F = 0.5

(d) F = 0.25

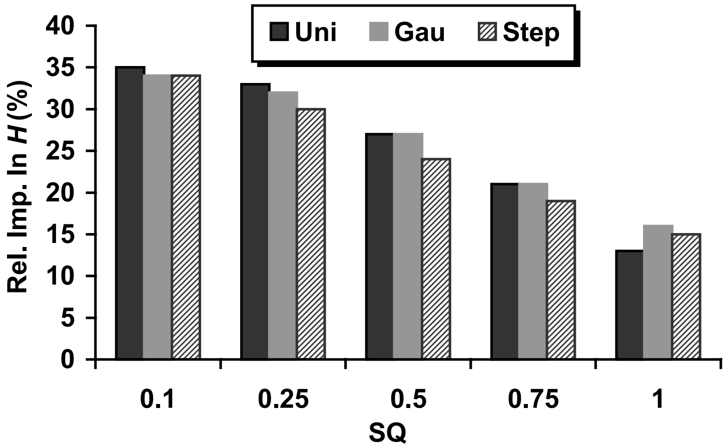
Figure 6.28 Performances of HUG for three distribution functions



(a) F = 1

(b) F = 0.75

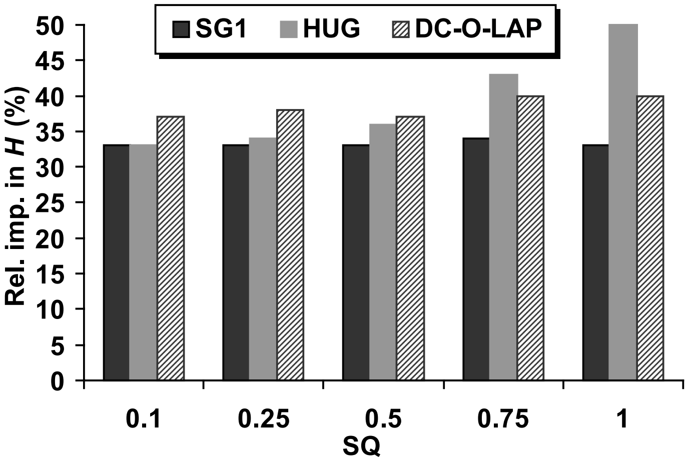
(c) F = 0.5



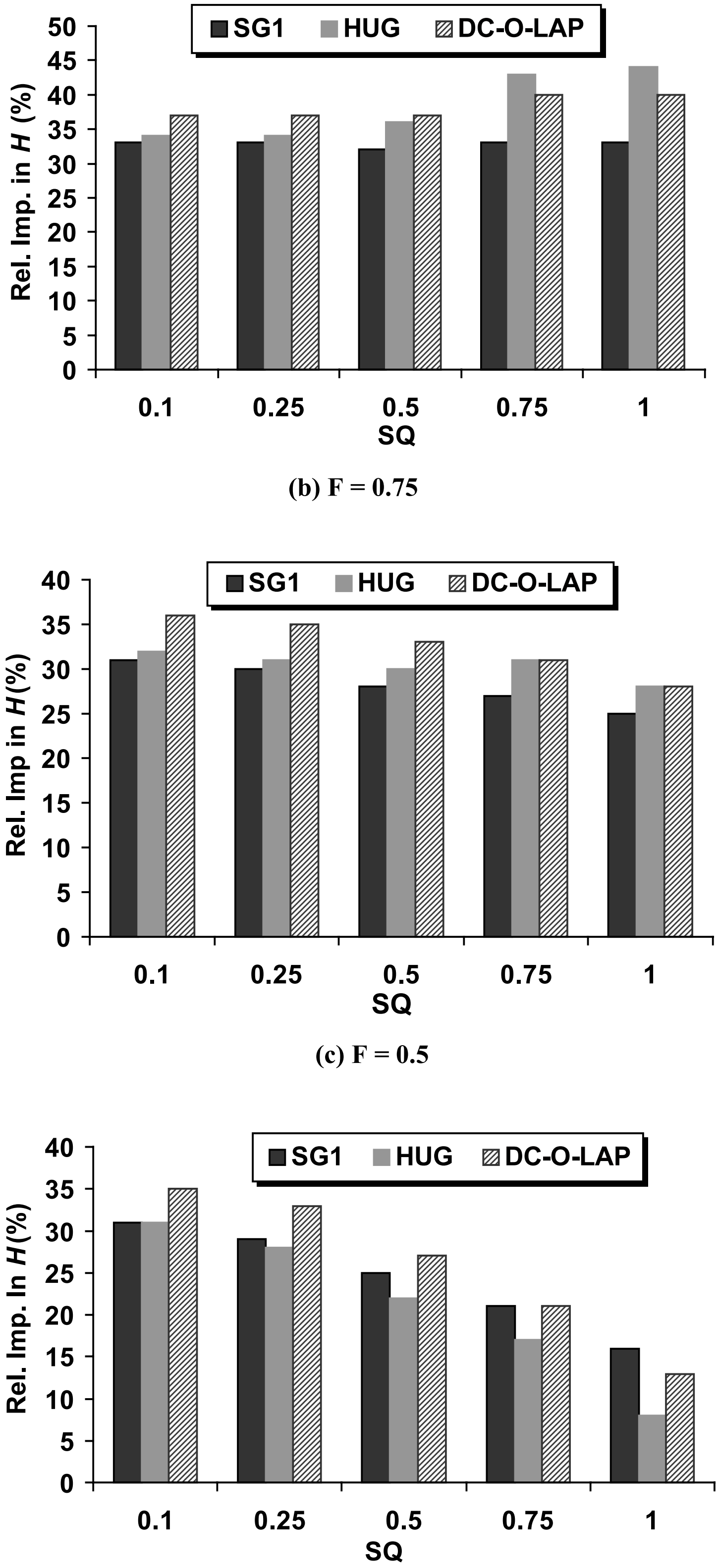
(d) F = 0.25

Figure 6.29 Performances of DC-O-LAP for three distribution functions

The Figure 6.30 and Figure 6.31 compare the three algorithms SG1, HUG, and DC-O-LAP for the workloads that are generated using the uniform function and the step function to model $\{F_{i,j}\}$ and $\{SQ_{i,j}\}$. The winners under different user model in terms of F and SQ is essentially the same as those shown in Table 6.3 for the workload generated using the Gaussian function.



(a) F = 1

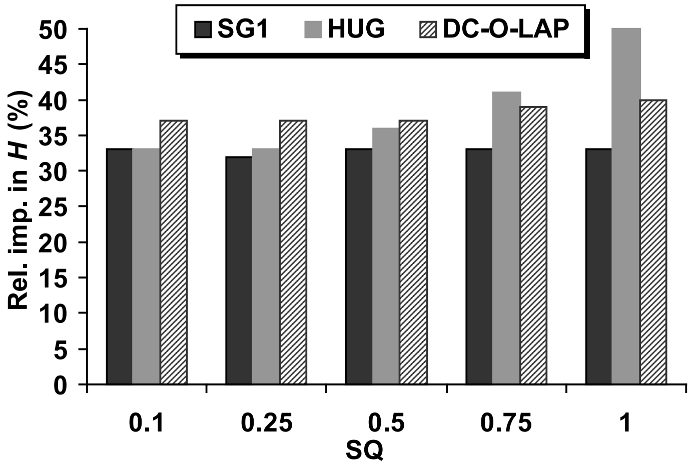


(b) F = 0.75

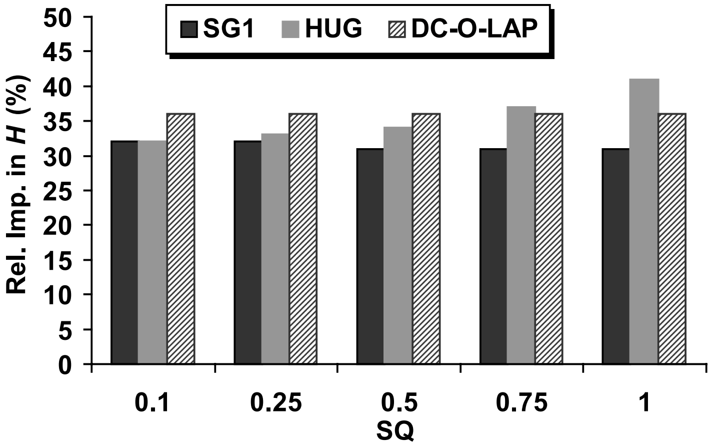
(c) F = 0.5

(d) F = 0.25

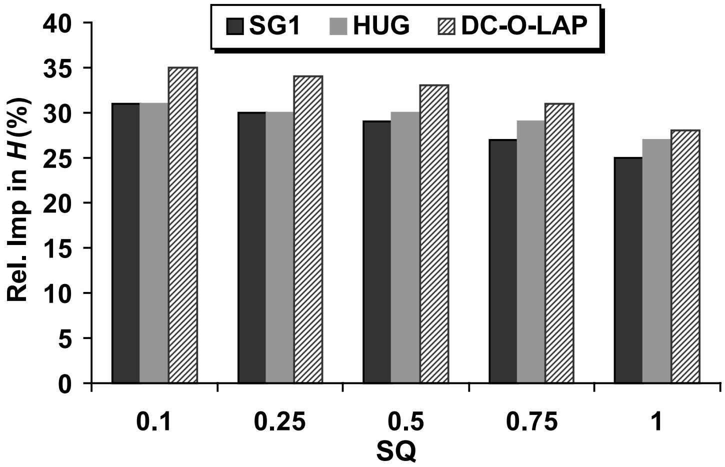
Figure 6.30 Performances of SG1, HUG and DC-O-LAP using uniform function to model {$F_{i,j}$} and {$SQ_{i,j}$}



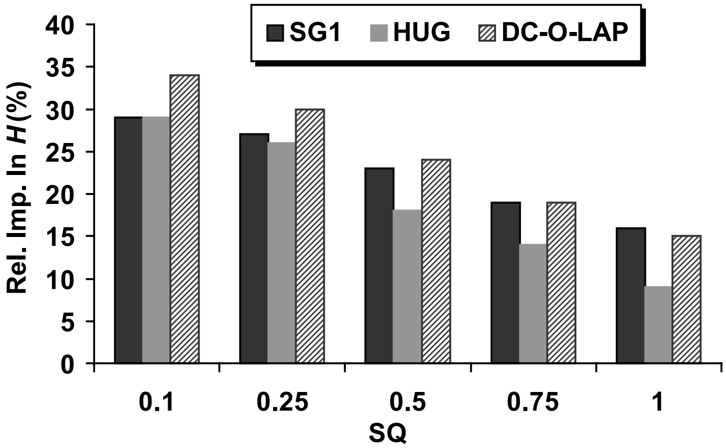
(a) F = 1



(b) F = 0.75



(c) F = 0.5



(d) F = 0.25

Figure 6.31 Performances of SG1, HUG and DC-O-LAP using stepwise function to model { $F_{i,j}$ **} and {** $SQ_{i,j}$ **}**

6.3 Summary of Results

(1) The content delivery approaches that combine publish-time and access-time placement based on subscription and access patterns provide much higher hit ratios in the local proxy servers than an access-based caching-only approach, which usually implies reduced user-perceived response time and smaller workload at the publishing sites. When F and SQ vary between 0.25 and 1, our best algorithms improve the hit ratio in local content servers by 33% to 133% for ALTERNATIVE trace, and by 16% to 50% for NEWS trace;

(2) In the publish/subscribe applications in which all requests are driven by matching and notifications, SG2 yields the best performance in most cases and introduces only comparable traffic burdens in the network over the caching-based approach;

(3) Under the combined user access model, HUG performs best when subscriptions fairly match users' actual access patterns; otherwise, Dual-Cache Partitioned by Operation with Limited Adaptive Partition (DC-O-LAP) is superior to other approaches. This is true for both traces we use in the simulation;

(4) For the trace with a more general popularity distribution of pages than that for a very busy news site, the benefits of the proposed approaches are more pronounced;

(5) Using the uniform, the stepwise and the Gaussian functions to model $\{F_{i,j}\}$ and $\{SQ_{i,j}\}$ does not affect our conclusions above.

Chapter 7

Conclusions

User provided information is a valuable resource for enhancing Internet services. This information is useful in applications ranging from user-facing content personalization and notification to infrastructural services. This thesis develops and studies two applications, one of which focuses on inferring online raters' reputations and enhancing application-level rating services, and the other addresses the use of subscription information in content delivery and caching in publish-subscribe middleware.

7.1 Building and Applying Reputation System in Rating Services

The study of building and using a reputation framework demonstrates the power of statistical analysis and data mining on raw user information, in this case ratings. The results demonstrate the feasibility of a fully automatic and self-maintained reputation framework for online rating management. While experimentally shown to match the criteria that human beings use for judging reputation, the presented algorithms outperform the manual determination by human beings, who are limited in their ability to process large and complicated information sources. In addition to the comparisons to the results of manual judgments, our experiments also demonstrate that our reputation system has the desirable properties such as the ability of reputations (and high-reputation raters) to differentiate items more finely.

7.1.1 CONTRIBUTIONS

This study has three primary research contributions:

1. Proposing a general framework for evaluating raters in rating-based communities;
2. Proposing a method to compute refined product scores by using reputation information;
3. Presenting a method to compute quantitative confidence in raters' qualification and objects' scores;
4. Evaluating the proposed reputation framework with the framework of a neutral online rating site;
5. Carrying out analyses of the characteristics of various types of online ratings.

This work may facilitate rating-aided applications in three aspects:

(1) Reputation information about raters can be used to identify experts or authorities in applications ranging from product/knowledge research to social networks;

(2) Value-added information about rated objects guides consumers in choosing products;

(3) The reputation scheme benefits site managers in better organizing site content, for example, ordering the reviews according to the reputations of the reviewers.

7.1.2 **Social Implications of** Reputation**-enhanced Rating Services**

In addition to aiding consumers, reputations also have some social implications:

(1) "Information explosion" vs. "rating scarcity"

The publication of reputations encourages content with high quality, providing a way to deal with both "information explosion" and "rating scarcity".

(2) Privacy vs. reliability

Some people do not want others to know their opinions on certain objects. This can be handled by hiding their identities and pegging them anonymously by reputation instead. Without a reputation scheme, anonymity often degrades the reliability of people's opinions, hence reputation information helps protect privacy without loss in reliability.

(3) Robustness of rating aggregation methods

Boosting scores by fake ratings is a typical cheating behavior in rating-based reputation schemes. With our reputation framework, intruders need to gain sufficient reputation to make their attack work. They will need to "attack" other rating profiles of items besides the product they are actually interested in, and understand how to gain high reputations based on our methods. Giving ratings (e.g. fake ratings) that deviate from those of the rating community leads to lower reputation, which means that such attacks must be conducted by not just a few but many conspirators (end users). Also, given that the rating sites are open to all users, as new ratings come in the rating profiles change, so maintaining a reputation artificially is a dynamic and expensive affair.

7.1.3 Future Work in Reputation-enhanced Rating Service

There are several directions to extend the study on rating services:

(1) Refining the methods to compute reputation by accounting for raters' temporal behavior

For example, let the importance of ratings decay with time, rather than giving all past ratings equal value at any given time as in the current work.

(2) Aggregating ratings from multiple sites

This rating service can be used in information integration from distributed rating sources. There are two directions to implement this aggregation. The first is to merge ratings given to the same item on multiple sites, and thus improving the confidence in scores of rated objects. Our current system can be easily adapted to this task. In this case, the same rater is viewed as different users at different sites.

Another direction is to aggregate reputation of one rater across multiple sites. Technically, only one layer needs to be inserted in the current reputation hierarchy. But eventually, this aggregation requires Internet-scale user identification which is not generally supported at present. One approach to deal with the problem may be to cooperate with services such as Microsoft .NET Passport [Microsoft] that supports single user identification across the Internet.

Because different web sites may use different category structures and cover different rated entities, one challenge of Internet-scale rating integration is to handle the heterogeneous category structures.

(3) Detecting fraudulent behaviors to boost reputations and/or product scores

Dishonest users may intentionally boost their reputations and eventually affect scores of products. One approach is to have several conspirators give the same ratings on the same products and thus earn high reputations. The approach may work if there are few ratings given by other users who do not participate in the conspiracy. One challenge is to identify the patterns of major cheating behaviors and to design methods for discovering these patterns effectively and promptly.

(4) Designing infrastructure to build and deploy a rating system

Rating information is growing quickly in a variety of areas as more and more people realize the importance of rating services. One major challenge in building rating engines is the scalability of the system, especially when ratings and reputations are aggregated from multiple distributed sites.

(5) Exploring the application of our reputation/rating mechanism in other areas

Rating service is needed in any scenario in which qualification information is important. One possible application of reputation/rating systems is to improve security and fault tolerance in Distributed Information Systems. For example, one important problem in loosely-coupled distributed information systems such as p2p networks is identifying a source's reliability. In p2p networks, each node exposes only limited information to other nodes. A reputation/rating system can be built to track the behavior of each node such as connection availability in the past and generate a rating for each node regarding its reliability in terms of availability. The replicating and searching algorithms in a p2p network can make use of this reliability rating of neighboring nodes to avoid querying unavailable nodes.

Our reputation system is applicable to the applications in which a decision is made based on multiple parameters and the importance of each parameter can be estimated from the performance of the parameter in the past. For example, reputation information can be used to build up novel collaborative recommendation systems. Existing collaborative recommendation methods usually rely on the similarity of users' opinions. We can improve the recommendation quality by incorporating the reliability information about each user.

7.2 Using Subscription Information for Enhancing Content Delivery

The major lesson learnt from the exploration of subscription-enhanced content delivery and caching algorithms is that user pre-defined interest is a valuable resource for identifying users' information needs on the fly. As an additional information source to user access patterns, subscription information can be used adequately for improving the hit ratio in the closest content servers, even when users' actual request patterns do not match the pre-provided subscriptions very closely.

7.2.1 Contributions

The major contributions of this study are as follows:

(1) Presenting the first study of content delivery and caching that uses publish-subscribe information when it is available;

(2) Proposing several content delivery and caching methods based on access type, subscription distribution, and access pattern, and comparing them under different scenarios to determine the best one;

(3) Experimentally demonstrating the benefit of our approaches in reducing the response time to end users without extra overhead in network traffic;

(4) Evaluating the tradeoff among the proposed mechanisms to understand what aspects are most important or are important;

(5) Developing realistic workloads for evaluation, including modeling the hybrid request sequence and the subscription quality;

(6) Building a simulator to study content delivery in globally distributed servers.

7.2.2 Future Work on Subscription-enhanced Content Delivery

The study on subscription-enhanced content delivery generates several interesting problems for further investigation:

(1) Extending evaluation functions by incorporating other factors regarding the importance of contents, e.g., the relations between contents can be used to associate related contents that match a group of subscriptions;

(2) Exploiting other opportunities for content replacement between the publishing time and the first request time, e.g., deliver a page from a publisher when the value of a published page is higher than some old pages in a cache;

(3) Introducing publisher-side content servers to deal with dynamic contents that can only be generated upon users' requests in database driven content sites;

(4) Designing content placement algorithms for cooperative content servers and making content placement engine collaborate with the matching engine and the routing engine in a content-based distributed publish/subscribe system;

(5) Deploying the proposed algorithms in a testbed of distributed computing systems such as Planetlab [Planetlab]. The nodes of Planetlab can act as content servers that serve users' requests in each geographical area.

7.3 Future Work on User Modeling

A major challenge in the study of algorithms and systems that rely on user information is a lack of real-world information and workloads. One can get real workloads from deployed ratings and notification services, or deploy the services oneself to generate workloads. Some researchers carry on small-scale user experiments that are usually limited in the samples selection, the experiment design, etc. Therefore, user modeling and the design of system simulator and workload generation are important for large-scale studies of web users' behavior. More research work can be done in this area.

Appendix A

Properties of Function to Build *Reputation* from *GM* and *GC*

1. Properties of the Function Used to Compute *Reputation*:

$$Reputation = f(GM,GC) = \left(GM + 1\right)^{GC} - 1$$

(1.1) $\partial f \big/ \partial GM = GC \bullet \left(GM + 1\right)^{GC-1}$

(1.2) $\partial f \big/ \partial GC = (GM + 1)^{GC} \bullet \log\left(GM + 1\right)$

(1.3) $\dfrac{\partial^2 f}{\partial GM \cdot \partial GC} = (GM + 1)^{GC-1} \bullet \left(1 + GC \bullet \log\left(GM + 1\right)\right)$

(1.4) $\dfrac{\partial^2 f}{\partial GC \cdot \partial GM} = (GM + 1)^{GC-1} \bullet \left(1 + \left(GM + 1\right) \bullet \log^2\left(GM + 1\right)\right)$

2. Proofs Showing That the Function Has the Desired Properties:

(2.1) It grows with *GM* faster if *GC* is bigger.

Proof:

$$\frac{\partial^2 f}{\partial GM \cdot \partial GC} = (GM + 1)^{GC-1} \bullet \left(1 + GC \bullet \log\left(GM + 1\right)\right)$$

$$\xrightarrow{GM,GC\in[0,1]} (GM + 1)^{GC-1} \geq \frac{1}{GM + 1} \geq 0 \; and \; \left(1 + GC \bullet \log\left(GM + 1\right)\right) \geq 1$$

$$\rightarrow \frac{\partial^2 f}{\partial GM \cdot \partial GC} \geq 0 \; for \; any \; GM \; and \; GC$$

(2.2) It grows with *GC* faster if *GM* is bigger.

Proof:

$$\frac{\partial^2 f}{\partial GC \cdot \partial GM} = (GM+1)^{GC-1} \bullet \left(1 + (GM+1) \bullet \log^2 (GM+1)\right)$$

$$\xrightarrow{GM,GC \in [0,1]} (GM+1)^{GC-1} \geq \frac{1}{GM+1} \geq 0 \; and \; \left(1 + (GM+1) \bullet \log^2 (GM+1)\right) \geq 1$$

$$\rightarrow \frac{\partial^2 f}{\partial GC \cdot \partial GM} \geq 0 \; for \; any \; GM \; and \; GC$$

(2.3) *GM* is the dominant variable if *GC* is big but *GM* is small. And *GC* is the dominant variable if *GM* is big but *GC* is small.

Proof:

$$\partial f \big/ \partial GM - \partial f \big/ \partial GC$$
$$= GC \bullet (GM+1)^{GC-1} - (GM+1)^{GC} \bullet \log(GM+1)$$
$$= (GM+1)^{GC-1} \bullet \left(GC - (GM+1) \cdot \log(GM+1)\right)$$

This function is shown in Figure A.1. From Figure A.1, some values along the z-axis are above 0, which means that Reputation grows with *GM* faster than it grows with *GC*, or in other words, *GM* is dominant. It happens when *GC* is big but *GM* is small. When *GC* is small but *GM* is big, the value of z-axis turns out to be negative, which indicates that *GC* is the dominant variable.

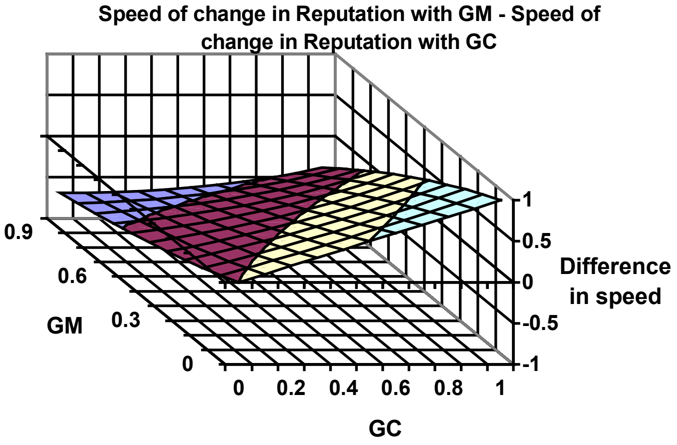


Figure A.1 $\frac{\partial \text{Reputation}}{\partial GM} - \frac{\partial \text{Reputation}}{\partial GC}$

Appendix B

Performance of Subscription-enhanced Content Delivery Algorithms for a Workload with Very Low First Time Miss Ratio

In this thesis, we study the performances of subscription-enhanced content delivery algorithms that are deployed at the edge content servers that serve users in a certain area. The first time miss ratio at such a server is usually about 40% as observed in real-world data. Our experiments in chapter 6 demonstrate the benefits of our algorithms by prefetching contents to reduce the first time miss ratio and by combining subscription and access information to decide a page's value accurately on the fly.

However, if our algorithms are applied at the content servers that are in front of a publisher, the first time miss ratio at those servers turns to be much lower than 40% because of high degree of concentration in users' requests at reverse proxy servers. In this case, the subscription-based approaches benefit content delivery mainly by evaluating pages more accurately than the caching approaches that are based on access pattern only. To study the benefits of our approach in this scenario, we generate a request workload using the methods to generate NEWS trace as discussed in chapter 5, except for increasing the total number of requests by one order of magnitude. Consequently, the first time miss ratio at the servers becomes 4.8%, which is much lower than that of NEWS trace with 34.2%.

To further understand the power of subscription information in predicting a page's left-over value on the fly, two variants of SG2 and SR are proposed. The two algorithms, designated as SG2-GD* and SR-GD* in this thesis, do not take the placement opportunity at publish-time and just use subscription information in cache-miss-driven content replacement.

Figure B.1 demonstrates the performance of the approaches when cache storage capacity is at 5% and 10% levels respectively. The performance is measured as miss ratio because the low first time miss ratio in the request trace results in high hit ratio using any approach.

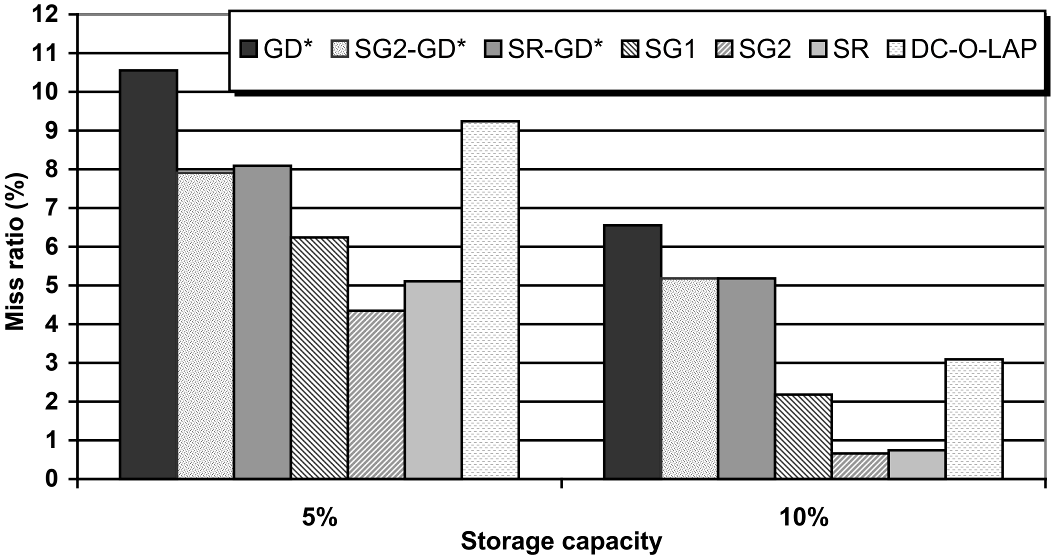


Figure B.1 Miss ratios of major algorithms when using a trace with first time miss ratio of 4.8%

Figure B.1 demonstrates that SG2-GD* and SR-GD*, which are different from GD* only in using subscription information in evaluation, yield lower miss ratio than GD*. Furthermore, SG1, SG2 and SR that place contents proactively at publish time are superior to the approaches that use on-demand access-time content placement only.

Interestingly, DC-O-LAP is not a good approach for the trace with low first time miss ratio. We conjecture that this is due to the fact that DC-O-LAP assigns a minimum of 25% storage to publish-time portion for storing new pages. However, when most accesses are repeated requests, it is better to assign more cache to store the pages that have been requested in DC-O.

Bibliography

[Akamai] Akamai. http://www.akamai.com.

[Amazon ’00] Amazon Inc. http://www.amazon.com/

[Altinel ’00] Altinel, M. and Franklin, M. J. Efficient Filtering of XML Documents for Selective Dissemination of Information. In Proceedings of 26th International Conference on Very Large Databases (VLDB 2000), 2000.

[Arkin] Arkin, H. and Colton, R. Statistical Methods, 5th edition, Barnes & Noble, Inc., 1970.

[Barford ’98] Barford, P. and Crovella, M. Generating Representative Workloads for Network and Server Performance Evaluation. In Proceedings of ACM Sigmetrics'98, 1998.

[Besravros ’95] Besravros, A. Demand-based Document Dissemination to Reduce Traffic and Balance Load. In Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing (SPDP’95), 1995.

[Bornhvd ’00] Bornhvd, C., Cilia, M., Liebig, C., and Buchmann, A. An Infrastructure for Meta-Auctions. In Proc. of Int'l. Workshop on Advanced issues of E-Commerce and Web-Based Information Systems (WECWIS), 2000.

[Breslau ’99] Breslau, L., Cao, P., Li, F., Phillips, G., and Shenker, S. Web Caching and Zipf-like Distributions: Evidence and Implications. In Proceedings of IEEE Infocom ’99, 1999.

[Brin ’98] Brin, S. and Page, L. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In Proceedings of WWW7 / Computer Networks 30(1-7): 107-117 (1998).

[BRITE] BRITE. http://www.cs.bu.edu/brite

[Cao ’97] Cao, P. and Irani, S. Cost-Aware WWW Proxy Caching Algorithms. In Proceedings of USENIX Symposium on Internet Technology and Systems, 1997.

[Cao ’04] Cao, F. and Singh J. P. Efficient Event Routing in Content-based Publish-Subscribe Service Networks", In Proceedings of IEEE INFOCOM 2004.

[Carzaniga ’98] Carzaniga, A., Rosenblum, D. S., and Wolf, A. L. Design of a Scalable Event Notification Service: Interface and Architecture. Tech. Rep. CU-CS-863-98, Department of Computer Science, Univ. of Colorado at Boulder, Sept. 1998.

[Chang ’02] Chang, C.-Y. and Chen, M.-S. Web Cache Replacement by Integrating Caching and Prefetching. Poster, CIKM 2002.

[Chi ’01] Chi, E. H., Pirolli, P., Chen, K., and Pitkow, J. Using Information Scent to Model User Information Needs and Actions on the Web. CHI 2001, Vol. 3, Issue 1, 490-497.

[Cidon ’01] Cidon, I., Kutten, S., and Soffer, R. Optimal Allocation of Electronic Content. In Proceedings of IEEE Infocom 2001.

[CNet ’04] CNET.com. http://www.cnet.com/

[CNN] CNN news site. http://www.cnn.com/

[Deolasee ’01] Deolasee, P., Katkar, A., Panchbudhe, A., Ramamritham, K., and Shenoy, P. Adaptive Push-Pull: Disseminating Dynamic Web Data. In Proceedings of WWW10, 2001.

[Dellarocas ’00] Dellarocas, C., Immunizing Online Reputation Reporting Systems Against Unfair Ratings and Discriminatory Behavior. In Proceedings of EC’ 00: The 2nd ACM Conference on Electronic Commerce, 2000.

[Digital Island] Digital Island. http://www.digitalisland.com/.

[Duchamp ’99] Duchamp, D. Prefetching Hyperlinks. In Proc. of USENIX Symp. on Internet Technologies and Systems, 1999.

[ebay] eBay Inc. http://www.ebay.com

[Epinions ’01] http://www.epinions.com

[Eugster ’00] Eugster, P. T., Guerraoui, R., and Sventek, J. Type-based publish/subscribe. Technical Report, Swiss Federal Institute of Technology, June 2000.

[Eugster ’03] Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A. M. The many faces of publish/subscribe. ACM Computing Surveys, Vol 35(2), pp. 114 – 131, 2003.

[Evans ’03] Evans, D., Wang, C., and Xie, J. A Payment Mechanism for Publish-Subscribe Systems. http://www.cs.virginia.edu/~evans/pubsub/

[Fabret ’00] Fabret, F., Llirbat, F., Pereira, J., and Shasha, D. Efficient Matching for Content-based Publish/Subscribe Systems. Technical report, INRIA'2000. http://citeseer.nj.nec.com/fabret00efficient.html

[Fabret ’01] Fabret, F., Jacobsen, H. A., Llirbat, F., Pereira, J., Ross, K. A., and Shasha, D. Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems, In Proceedings of SIGMOD 2001.

[Fan '99] Fan, L., Cao, P., Lin, W., and Jacobson, Q. Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance. In Proceedings of ACM SIGMETRICS, 1999.

[Fitzpatrick '00] Fitzpatrick, G., Kaplan, S, Mansfield, T., Arnold, D., and Segall, B. Supporting public availability and accessibility with Elvin: Experiences and Reflections. 2000. http://elvin.dstc.edu.au/doc/papers/jcc/jcc.pdf

[Gadde '00] Gadde, S., Chase, J., and Rabinovich, M. Web Caching and Content Distribution: A View From the Interior. 5th International Web Caching and Content Delivery Workshop (WCW '00), 2000.

[Gryphon '98] Strom, R., Banavar, G., Chandra, T., Kaplan, M., Miller, K., Mukherjee, B.,

Sturman, D., and Ward, M. Gryphon: An Information Flow Based Approach to Message Brokering. International Symposium on Software Reliability Engineering '98 Fast Abstract.

[GryphonM '99] Aguilera, M. K., Strom, R. E., Sturman, D. C., Astley, M., and Chandra, T. D. Matching events in a content-based subscription system. In Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing.

[GryphonR '99] Banavar, G., Chandra, T., Mukherjee, B., Nagarajarao, J., Strom, R. E., and Sturman, D. C. An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems. In Proceedings of 19th IEEE International Conference on Distributed Computing Systems.

[GryphonR '00] Opyrchal, L., Astley, M., Auerbach, J., Banavar, G., Strom, R., and Sturman, D. Exploiting IP Multicast in Content-Based Publish-Subscribe Systems. In Proceedings of Middleware 2000.

[GryphonC '03] Pietzuch, P. and Bhola, S. Congestion Control in a Reliable Scalable Message-Oriented Middleware. In Proceedings of Middleware 2003.

[GryphonD ’03] Sumeer Bhola, Yuanyuan Zhao, Joshua Auerbach. Scalably Supporting Durable Subscriptions in a Publish/Subscribe System. In Proceedings of International Conference on Dependable Systems and Networks 2003.

[Guernsey ’00] Guernsey, L. Suddenly, Everybody's an Expert on Everything. The New York Times, Feb. 03, 2000.

[Gwertzman ’97] Gwertzman, J. and Seltzer, M. An Analysis of Geographical Push-Caching. 1997.

[Hafner ’01] Hafner, K. Web Sites Begin to Self Organize. The New York Times, Jan. 18, 2001.

[Half] half.com by ebay. http://half.ebay.com

[Heimbigner ’00] Heimbigner, D. Adapting Publish/Subscribe Middleware to Achieve Gnutella-like Functionality. In Proc. of the 16th ACM Symposium on Applied Computing (SAC'2001) 2001.

[Hill ’95] Hill, W., Stead, L., Rosenstein, M. and Furnas, G. Recommending and evaluating choices in a virtual community of use. In Proceedings of the 1995 ACM Conference on Human Factors in Computing Systems (1995). ACM, New York, pp. 194-201.

[Huang ’01] Huang, Y.-Q. and Garcia-Molina, H. Publish/Subscribe in a Mobile Environment. MobiDE 01.

[Jin ’01] Jin, Shudong and Bestavrous, A. GreedyDual* Web Caching Algorithm: Exploiting the Two Sources of Temporal Locality in Web Request Streams. Computer Comm., vol. 24(2), pp. 174-183, Feb. 2001.

[Kangasharju ’01] Kangasharju, J., Roberts, J., and Ross, K. W. Object Replication Strategies in Content Distribution Networks. In Proceedings of WCW'01: Web Caching and Content Distribution Workshop, Boston, MA, June 2001.

[Karger ’97] Karger, D., Lehman, E., Leighton, F. T., Levine, M., Lewin, D., and Panigrahy, R. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In Proceedings of the 29th Annual ACM Symposium on Theory of Computing, pages 654--663, 1997.

[Karlsson ’02] Karlsson, M. and Mahalingam, M. Do We Need Replica Placement Algorithms in Content Delivery Networks. In Proc. Of WCW ’02, 2002.

[Kleinberg ’99] Kleinberg, J. Authoritative sources in a hyperlinked environment. Journal of the ACM 46 (1999).

[Konstan ’97] Konstan, J. A., Miller, B. N., Maltz, D., Herlocker, J. L., Gordon, L. R., and Riedl, J. GroupLens: Applying Collaborative Filtering to Usenet News. Communications of ACM, Vol. 40, No. 3, 77-87, March 1997.

[Korupolu ’02] Korupolu, M. R. and Dahlin, M. Coordinated Placement and Replacement for Large-Scale Distributed Caches. IEEE Transactions on Knowledge and Data Engineering, 2002 (Vol. 14, No. 6), pp. 1317-1329.

[Krishnan ’00] Krishnan, P., Raz, D., and Shavitt, Y. The cache location problem. IEEE/ACM Transactions on Networking, 8(5): pages 568-582, October 2000.

[Li, B. ’99] Li, B., Golin, M. J., Italiano, G. F., and Deng, X. On the Optimal Placement of Web Proxies in the Internet. In Proceedings of IEEE INFOCOM’99, 1999.

[Li, D. '99] Li, D. and Cheriton, D. R. Scalable Web Caching of Frequently Updated Objects Using Reliable Multicast. In Proceedings of USENIX Symposium on Internet Technologies and Systems, 1999.

[Lieberman '95] Lieberman, H. Letizia: An Agent That Assists Web Browsing. Proceedings of the 1995 International Joint Conference on Artificial Intelligent, 1995.

[Makwana '02] Makwana, D. An Introduction to Semantic Networking and Its Performance Evaluation. MS Thesis, Rutgers, The State University of New Jersey. January 2002. http://www.caip.rutgers.edu/~makwana/Acads/Research/thesis.pdf

[Microsoft] Microsoft .Net Passport. http://www.passport.net/

[MSNBC] MSNBC. http://www.msnbc.com/news

[Padmanabhan '00] Padmanabhan, V. N. and Qiu, L.-L. The Content and Access Dynamics of a Busy Web Site: Findings and Implications. In Proceedings of ACM SIGCOMM 2000.

[Pastry '01] Rowstron, A. and Druschel, P. Pastry: Scalalble, decentralized object locating and routing for large-scale peer-to-peer systems. In Proceedings of Middleware 2001.

[Planetlab] www.planet-lab.org

[Qiu '01] Qiu, L., Padmanabham, V. N., and Voelker, G. M. On the placement of web server replicas. In Proceedings of 20th IEEE INFOCOM, 2001.

[Rabinovich '98] Rabinovich, M. Issues in Web Content Replication. http://citeseer.nj.nec.com/rabinovich98issues.html

[Resnick ’94] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In Proceedings of CSCW’94, pp.175-186, Chapel Hill, NC.

[Riggs ’01] Riggs, T. and Wilensky, R. An Algorithm for Automated Rating of Reviewers. In Proceedings of JCDL’01, pp. 381 – 387, 2001.

[Saroiu ’02] Saroiu, S., Gummadi, K. P., Dunn, R. J., Gribble, S. D., and Levy, H. M. An Analysis of Internet Content Delivery Systems. In Proc. of OSDI ’02, 2002.

[Scribe ’01] Rowstron, A., Kermarrec, A. M., Castro, M., and Druschel, P. Scribe: The design of a large-scale event notification infrastructure. In Proceedings of 3rd International Workshop on Networked Group Communication (NGC), 2001.

[Skarmeas ’98] Skarmeas, N. and Clark, K. L. Content based Routing as the Basis for Intra-agent Communication. In Proceedings of ATAL-98, (ICMAS'98), 1998.

[Tang ’00] Tang, W.-T. and Mutka, M. W. Intelligent browser initiated server pushing. In Proc. of the IEEE Intern. Performance, Computing, and Communications Conference, 2000.

[Terveen ’97] Terveen, L., Hill, W., Amento, B., McDonald, D. and Creter, J. PHOAKS: A System for Sharing Recommendations. Communications of ACM, Vol. 40, No. 3, 59-62, March 1997.

[TIBCO ’99] TIB/Rendezvous. White Paper. TIBCO, Parlo Alto, CA.

[Venkataramani ’01 TR] Venkataramani, A., Yalagandula, P., Kokku, R., Sharif, S., and Dahlin, M. The Potential Costs and Benefits of Long-term Prefetching for Content Distribution. Technical Report TR-01-13, UT, Austin, 2001.

[Venkataramani ’01] Venkataramani, A., Weidmann, P., and Dahlin, M. Bandwidth Constrained Placement in a WAN. In Proceedings of ACM Symposium on Principles of Distributed Computing, 2001.

[Wang ’99] Wang, J. A Survey of Web Caching Schemes for the Internet. ACM Computer Communication Review, 29(5):36--46, October 1999.

[Wang ’02] Wang, C., Carzaniga, A., Evans, D., and Wolf, A. L. Security Issues and Requirements for Internet-Scale Publish-Subscribe Systems. In Proc. of Hawaii International Conference on System Sciences, 2002.

[Wang ’02] Wang, L.-M., Pai, V., and Peterson, L. The Effectiveness of Request Redirection on CDN Robustness. In Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI'02), 2002.

[Wolman USITS ’99] Wolman, A., Voelker, G., Sharma, N., Cardwell, N., Brown, M., Landray, T., Pinnel, D., Karlin, A., and Levy, H. Organization-Based Analysis of Web-Object Sharing and Caching. In Proceedings of 2nd USENIX Symp. on Internet Technologies and Systems, 1999.

[Wolman SOSP ’99] Wolman, A., Voelker, G., Sharma, N., Cardwell, N., Karlin, A., and Levy, H. On the Scale and Performance of Cooperative Web Proxy Caching. In Proceedings of the 17th ACM Symposium on Operating Systems Principles, 1999.

[Yahoo] Yahoo Inc. http://www.yahoo.com/

[Zacharia ’99] Zacharia, G., Moukas, A. and Maes, P. Collaborative Reputation Mechanisms in Electronic Marketplaces. In Proceedings of the 32nd Hawaii International Conference on System Science, January 5-8, 1999.

Publications

1. Mao Chen, J. P. Singh, and Andrea LaPaugh. *Subscription-enhanced Content Delivery*. In Proceedings of 8th International Workshop on Web Content Caching and Distribution, 2003.
2. Mao Chen, Andrea LaPaugh, and J. P. Singh. *Content Distribution for Publish/Subscribe Services.* In Proceedings of ACM/IFIP/USENIX International Middleware Conference 2003, pp. 83 – 102.
3. Mao Chen, Andrea LaPaugh, and J. P. Singh. *Categorizing Information Objects from User Access Pattern*. In Proceedings of 11th ACM International Conference on Information and Knowledge Management (CIKM 2002), 2002, pp. 365 – 372.
4. Mao Chen, Andrea LaPaugh, and J. P. Singh. *Predicting Category Accesses for a User in a Structured Information Space*. In Proceedings of the 25th ACM SIGIR, 2002, pp. 65 – 72.
5. Mao Chen and J. P. Singh. *Computing and Using Reputations for Internet Ratings*. In Proceedings of the 3rd ACM Conference on Electronic Commerce, 2001, pp. 154 – 162.
6. Minwen Ji, Edward W. Felten, J. P. Singh, and Mao Chen. *Managing Dynamic Content for Cluster-Based Application Servers*. Technical Report TR-621-00, Department of Computer Science, Princeton University, May 2000.