URL-tree와 URL-net를 사용한 인터넷 트래픽 분석

○
안광림, 김기창,
인하대학교 전자계산공학과

Web Traffic Analysis using URL-tree and URL-net

Kwang-Rim Ahn, Ki-Chang Kim
Dept. of Computer Science & Engineering INHA Univ.

인터넷 사용의 증가로 인한 정체 현상을 극복하기 위한 방안으로 캐시 서버를 사용하고 있다 캐시 서버의 보다 효율적인 사용을 위해 해당 네트웍의 트래픽에 대한 이해는 매우 중요하다. 즉 보다 적극적인 캐싱 전략을 수립하기위해 트래픽 분석이 선행되어야 한다. 본 논문에서는 URL-tree와 URL-net이라는 자료구조를 제안하고, 이것을 이용하여 웹 트래픽 분석을 수행한다 이러한 자료구조를 통해 웹 트래픽에 존재하는 '참조의 연결성'이라는 성질을 찾을 수 있다 본 논문에서는 위의 두 자료구조들이 인터넷 트래픽을 분석하는데 어떻게 도움을 주고 그러한 분석이 효율적인 캐싱 전략을 수립하는데 어떻게 사용될 수 있는가를 보여준다.

1. 서론

인터넷 교통량의 폭발적인 증가에 대해서는 많은 보고서가 그 현황을 보고하고 있다 [Kwa95]는 NCSA 웹 서버에 대한 조회가 94 년 2월부터 같은 해 12 월까지 3-4 개월마다 1,000,000 조회씩 증가한다는 것을 보고하고 있는데 현재 NCSA 서버는 평균 하루에 3,000,000 조회가 이루어지고 있는 것으로 알려져 있다. [Alo92]는 이러한 인터넷 교통량이 적절한 캐시 사용에 의해 60-90% 정도 경감될 수 있다는 예측을 trace-driven 시뮬레이션에 의해 보여주고 있다. 또한 [Dan93]도 NSF NET 교통량에 대한 분석에서 30-50% 정도의 교통량이 읽기 전용의 FTP 파일들에 대한 반복 접근임을 밝히고 따라서 캐시 사용에 의해 50% 정도의 파일 전송을 줄일 수 있다고 예측하고 있다. 이러한 교통량 분석을 토대로 인터넷 캐시에 대한 연구가 각국에서 활발히 진행되고 있다

이러한 캐싱 프락시 서버들의 성능을 향상 시키기 위해서는 보다 효율적인 캐싱 전략이 필수적인 요소이다 따라서 이러한 캐싱 전략을 수립하기 위해서는 해당 네트웍 사용자들의 웹 사용 성향을 파악하는 트래픽 분석이 선행 되어야 한다.

본 논문에서는 URL-tree, URL-net이라는 자료구조를 제안하고 이를 바탕으로 인터넷 트래픽을 분석한다. 또한 이로부터 얻어지는 일체적인 정보 또는 참조의 연결성(Reference Pattern) 등을 이용하여 보다 효율적인 캐싱 전략을 제안하고 구현한다.

2장에서는 URL-net 유사 연구와 기존 트래픽 분석, 3장에서는 URL-tree와 URL-net의 개념을 설명한다. 4장 5장에서는 인터넷 트래픽에 대한 static, dynamic 분석을 보이며, 6장 7장에서는 위에서 얻은 결과를 캐싱 전략에 이용하는 것과 결론을 말한다.

2. 트래픽 분석 Survey, URL-net 유사 연구

웹 트래픽 분석은 분석 데이터를 모으는 장소에 따라 Server, Client, Proxy의 세 가지 형태로 나눌 수 있다. 첫째, 서버측에서의 분석[Igo][Han]은 하나의 웹 서버에 접근하는 사용자들의 요구를 분석하게 된다. 둘째, 클라이언트측 분석[Car95]은 특정 사용자 그룹을 대상으로 웹 사용 시 마다 사용 내용을 저장케 하여 나중에 모아서 분석을 하게 된다. 이 방법은 각 사용자의 특성을 분석할 수 있는 장점이 있으나 대상 사용자의 웹 브라우저를 변경해야 하므로 많은 사람을 대상으로 할 수 없다는 단점이 있다. 마지막 프락시 측에서 분석하는 방법이 현재 가장 많이 연구되고 있는 분야로서, 프락시는 여러 사용자와 웹 서버 사이에 존재하므로 많은 사용자들의 요구를 분석할 수 있다. 다음 표는 기존 프락시 분석[Gha]에서 얻을 수 있는 일반화된 특징들이다.

표 2-1

Number	Name	Description
1	Median file size	≈ 2K
2	Mean file size	Less than 27K
3	File Types (accesses)	90-98% of accessed files are of types Graphics, HTML, and ftp-data
4	File Types (bytes)	Most bytes accessed are of type Graphics
5	% of accesses to unique Servers	Less than 11%
6	% of servers referenced one time	Less than 5%
7	Accesses concentration (servers)	25% are responsible for 80-95%
8	Bytes concentration (servers)	25% are responsible for 90% of the bytes accessed
9	Success Rate	88-99%
10	Self-similarity	$0.59 < H < 0.94$

위와 같은 기존의 트래픽 분석 방법들은 비교적 제한된 영역의 사용자들을 대상으로 평면적으로 분석한다. 이에 반해 본 연구는 패킷 스니퍼링(Packet Sniffering)을 이용하여 서브 네트웍 내의 모든 웹 사용자를 대상으로 하고 있으며, 몇 가지 자료구조를 이용하여 입체적인 분석과 캐싱 전략 수립에 도움이 되는 정보를 제공한다.

[Ven]에서는 웹 서버에 접근하는 사용자의 패턴을 분석하여 prefetch에 이용한다. 웹 서버는 미리 가지고 있는dependency graph라는 구조의 사용자 access pattern 분석 자료를 바탕으로 사용자가 특정 URL 페이지를 보는 동안 그 다음 접근될 가능성이 높은 페이지들을 client가 prefetch할 수 있도록 한다. 이 방법은 사용자의 access pattern을 이용하는 하나의 방법으로 client에게 prefetch에 필요한 정보를 전해주는 piggyback기능이 필요하다

3. URL-tree, URL-net 개념, example

URL-tree는 URL들의 디렉토리 구조를 반영하는 자료구조 이다. URL-tree를 구축함으로써 캐시 서버는 현재 접근되고 있는 URL들에 대한 일체적인 정보를 얻을 수 있다. 예를 들어

http //www x1/x2/x3
http //www.x1/x2/x4
http://www y1/y2/y3
http //www.x1/x2/x4
http://www.y1/y2/y3

와 같은 참조 리스트는 아래 그림3-1의 URL-tree를 형성하게 될 것이다

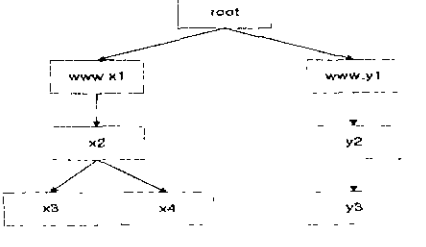


그림3-1. URL-tree 예

이 트리 상의 각 노드들에 페이지 크기, 참조 횟수, 참조 시간 등의 정보를 기억 시킨다. 예를 들어 위에서 x4와 y3 노드는 참조 횟수가 2가 될 것이다 이 URL-tree의 모양을 일, 주, 월 단위로 그 변화 모양을 분석하여 어느 부분이 얼마나 자주 어느 정도 변화하는가 등에 관한 정보를 분석한다 주기적인 변화 패턴은 말단 노드에서만 검색될 수도 있고 여러 노드를 포함하는 서브 트리에 대해서도 검색될 수 있을 것이다. 주기적인 변화 패턴을 보이는 노드 혹은 노드집합은 캐싱의 단위로 사용이 될 수 있을 것이다 예를 들어 prefetch 혹은 캐시 교체시 캐시로부터의 제거 혹은 삽입의 단위로 사용이 될 수 있을 것이다. 또한 병렬 캐시를 운영하는 경우 URL-tree로 부터 얻어지는 임체적인 정보를 load분산에 적용할 수 있다. URL-net는 위 트리 상에 간선(edge)을 부여함으로써 얻어진다. URL-net는 각 URL마다 그 페이지를 참조한 URL을 HTTP 패킷의 referrer 필드에서 얻어내어 부모 URL로부터 자식 URL에 간선을 첨가함으로써 얻어진다. 예를 들어 x4를 참조하던 사용자가 y3로 점프한다면 x4로부터 y3에 간선이 첨가될 것이다 x4의 HTML 파일에 y3 와 www.y1이 참조링크로 포함되어 있었다면 사용자가 www y1으로 점프하는 경우도 있을 것이다 이러한 사건이 발생하면 x4로 부터 www.y1에도 간선이 첨가된다. 어느 한 노드에서 출발하는 간선들은 분기확률을 가지게 된다. x4의 경우 사용자가 70'30의 비율로 y3와 www.y1으로 각각 점프한다면 간선(x4,y3)의 분기확률은 0.7이고 간선 (x4,www.y1)의 분기확률은 0.3이 될 것이다 그림3-2는 x4에 대해 분기확률을 표시한 URL-net의 한 예이다 캐싱된 모든 URL 노드의 모든 참조 링크에 대해 분기확률을 기억하고 갱신하는 것은 캐시 서버에 대한 지나친 부하가 될 것이다. 하지만, 주요 참조 링크에 대해 이러한 정보를 기억하고 있는 것은 캐시 서버의 캐싱 전략에 큰 효율성을 부여할 것이다

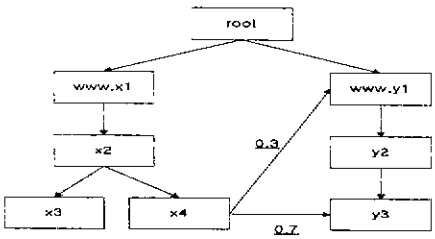


그림3-2. URL-net 예

URL-tree와 URL-net가 제공하는 정보는 세가지 면에서 이용된다. 첫째는 캐시 적중이 일어났을 경우이다 캐시 적중이 일어나면 캐시 서버는 그 다음에 어떤 URL들이 참조될 것이라는 것을 URL-net로부터 확률적으로 예측할 수 있다. 확률이 높은 URL페이지들을 미리 메모리에 저장해 놓음으로써 응답 시간을 감소 시킬 수 있다 두번째는 캐시 교체의 경우이다. 캐시 교체의 기본 단위는 URL 참조 패턴이 된다. 동일한 패턴에 속하는 URL 페이지들의 일부만을 캐시 안에 불러 들이는 것은 캐시 미스를 일으킬 확률을 높이기 때문에 피해야 한다. 마지막은 URL 페이지 자동 갱신의 경우이다. 캐시된 각 URL 페이지들이 가장 최신에 갱신된 정보를 담고 있도록 유지해야 하는 것은 캐시 서버의 중요한 기능 중의 하나이다. 자동갱신은 인터넷 트래픽을 증가시키므로, 가능한 한 최소화해야 할 것이다. URL-tree가 각 노드들의 변경주기를 반영하고 있으므로, 이에 근거하여 자동 갱신이 수행된다.

4. Static 분석

인터넷 트래픽 데이터를 얻기 위해 약 4개월 간 인하대학교의 한 세그먼트에 대해 트래픽 데이터를 수집하였다. 데이터는 tcpdump에 의해 모든 패킷의 처음 300 바이트들이 추출되었으며(매일 약 20 MB정도), 이 데이터들로부터 우선 http GET 패킷들을 추출하여 URL 리스트를 작성하였다. 이 URL 리스트 상의 각 URL들에게 우선 http HEAD 패킷을 보내어 페이지의 크기와 종류, 마지막 변경 시각 등의 정보를 추가하였다. URL 리스트에 있는 URL마다 다음과 같은 정보를 기록한다.

```
{ src : 클라이언트의 IP
  dst  서버의 IP
  URL  URL 스트링
  req_t: 참조시간
  ans_t  응답시간
  [illegible] 페이지 크기
  [illegible] 이 URL에 대한 지난 번 참조 [illegible] 이 URL의 내용이 변했는가 여부
  mf_scale: 변화정도
  rfn      : 참조횟수( 누적 )
  mf_freq  변화주기
  loc:     사이트 위치, 교내, 국내, 혹은 국외
}
```

이러한 형태의 데이터를 분석한 결과는 표4-1로 요약할 수 있다

표4-1

	Access	Access (%)	Bytes	Byte (%)	Mean Size
Total	347183	100	5020272167	100	14460
Image	167871	48.4	1372022863	27.2	8173
Text	145872	42.0	497480013	10 0	3410
Audio	1435	0 4	1111551977	22 1	774600
Video	85	0.0	171992680	3 4	2023443
Etc	31920	9 2	1867224634	37.2	58497
Accesses to unique servers : 11584 (0.4 %)					
Accesses to unique URLs : 416759 (14 2 %)					

기존의 분석 자료보다 이미지 데이터의 byte 비율이 약간 낮아지고 대신 audio, video, etc가 높아졌다. 이는 웹 사용이 고급화 되면서 java기술 또는 audio, video등의 멀티미디어 데이터에 대한 관심이 높아졌기 때문으로 여겨진다. Uniqt server와 Unique URL에 대한 요구 횟수는 각각 11584번, 416759번으로 전체에서 0.4%, 14 2%를 차지한다. 이는 참조의 지역성이 매우 높음을 나타낸다. 즉 캐싱을 함으로써 많은 이익을 가져올 수 있음을 알 수 있다.

5. Dynamic 분석

squid는 가장 많이 사용되고 있는 웹 프락시 캐시 서버이다 Harvest 캐시 서버로부터 발전되었으며 ICM 조사[ICM]에 의하면 현재 유럽의 캐시 사용자중 약 70%를 차지하고 있다. 기존의 캐싱 소프트웨어와는 달리 모든 요구를 하나의 non-blocking, I/O-driven process로 처리하며, 계층적 구조를 이용하므로 높은 성능을 보이고 있다.

squid는 요구되는 하나의 URL에 대한 정보를 StoreEntry라는 구조체로 표현한다. URL 요청에 대해 캐시에 있다면 그 정보를 client에게 보내주게 되고, 없다면 remote server에서 가져온 정보를 client에게 보내주게 된다. 이때 ClientBuildReplyHeader라는 함수를 통해 client에게 보낼 메시지의 header를 만들게 된다.

모든 요구의 처리가 이 부분을 통과 하므로 URL-tree/net을 생성, 변경하는 부분을 여기에 삽입하게 된다 URL-tree와 URL-net의 노드는 UtnNode라는 구조체로 표현한다. UtnNode의 정의는 다음과 같다.

```
struct _UtnNode {
                        StoreEntry *se,
                                int referer,
                                u_short ref_cnt
                                char name[256],
                                ssize_t size,
                                time_t timestamp;
                                time_t lastref,
                                time_t lastmod,
                                char
                        type[HTTP_REPLY_FIELD_SZ];
                                int has_url,
                                int ref_frq;
                                int change_frq,
                                int next_sibling,
                                int next_child,
                                int pattern[100],
};
```

squid의 ClientBuildReplyHeader에 ClientProcessUtnNode라는 함수를 삽입함으로써 각 요구에 대한 응답 메시지를 만들 때 마다 UtnNode관련 작업을 호출하게 한다. UtnNode를 만들고 갱신하여 URL-tree/net을 구성하는 알고리즘은 다음과 같다

알고리즘 5-1 clientProcessUtnNode

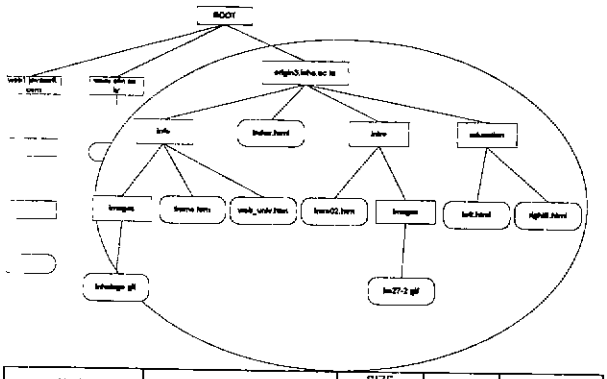
```
Input : * StoreEntry
Ouput : void

Step1. StoreEntry에 해당하는 URL을 분석하여 UtnNode의 단위만큼씩
잘라 처리한다. ( looping 1 ~ 3 )
Step2  해당 UtnNode가 존재하는지 검사한다. (searchUtnNode())
Step3.  없다면  생성( clientCreateUtnNode ),  존재한다면
갱신( clientUpdateUtnNode )한다.
```

Step1에서는 주어진 StoreEntry에 있는 URL 정보를 가지고 loop을 [illegible] 예를 들어 URL이 http://www/x/y 일 경우 UtnNode는 http://www, http://www/x, http://www/x/y 의 세 가지가 만들어진다 이

각각이 한번의 loop에서 처리된다. loop의 내용인 step2, step3에서는 우선 처리하고자 하는 UtnNode가 이미 존재하는지 검사한다. 검사 후 존재하지 않는다면 ClientCreateUtnNode를 호출하여 해당 UtnNode를 생성하고, 이미 존재한다면 ClientUpdateUtnNode를 호출하여 그 Node의 정보를 적절히 갱신한다.

사용자가 origin3.inha ac.kr를 비롯한 사이트들을 둘러 보았을 경우를 가정했을 때, squid가 만들어내는 URL-tree와 Node table은 그림5-1 에, URL-net은 그림 5-2, URL-net으로 부터 뽑아낸 몇 개의 참조 패턴은 그림 5-3에 보여진다.



Node	URL	SIZE (byte)	접근횟수	소속 URL 수
origin3 inha.ac kr	-	42049	247	6
info		24414	157	2
index html	Http://origin3 inha ac kr/index html	5116	4	0
intro		10305	36	2
education		2214	26	1
images		10522	62	1
web_univ htm	Http //origin3 inha ac.kr/info/web_univ htm	13892	1	0
frame학교생활 htm	Http //origin3.inha ac kr/intro/frame학교생활 htm	207	6	0
images	-	10098	30	1
left htm	Http //origin3.inha ac kr/education/left.html	2214	1	0
inhalogo gif	Http //origin3 inha ac kr/Info/images/inhalogo gif	10522	6	0
ba27-2.gif	Http://origin3 inha ac kr/intro/images/ba27-2.gif	10098	1	0

그림5-1. URL-tree와 Node table

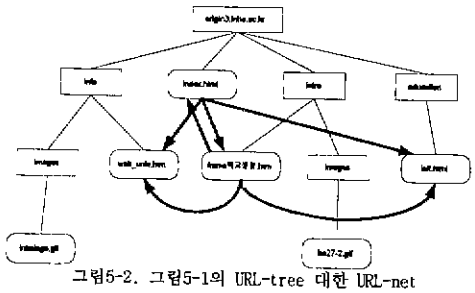


그림5-2. 그림5-1의 URL-tree 대한 URL-net

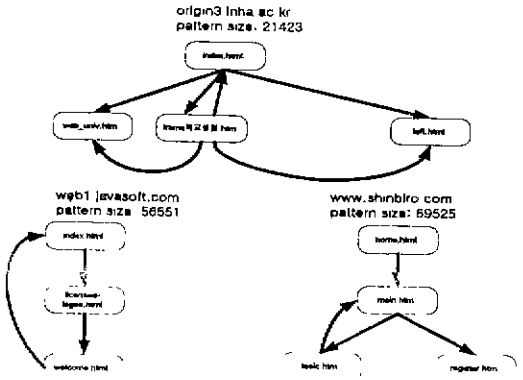


그림5-3. URL 참조 패턴

UtnNode 구조체로 URL-tree/net을 구현할 때 편의상 squid 시작 시 구조체 배열을 만들어 사용하였으며, 노드간의 포인팅은 구조체 배열의 인덱스를 사용하였다.

6. 캐시 서버에서의 '참조의 연결성' 이용

squid는 캐시 교체 알고리즘으로 다음과 같은 방법을 사용한다. Storage LRU Expiration Age라는 값을 사용하여 일정 시간동안 접근되지 않는 object를 무조건 지워버린다. 이 값은 저장공간(object의size)을 기준으로 동적으로 계산된다. Config file에서 값을 설정할 수 있으며 디폴트 값은 한 달 이다 이 값이 0이면 disk full 까지 캐시에 보존하게 한다. squid는 storeMaintainSwapSpace함수를 main loop에서 주기적으로 호출한다. 그 안에서는 storeCheckExpired함수를 이용하여 캐시안의 object들중 expire된 것을 찾아 storeRelease함수를 호출하여 cache에서 지워버린다. 본 연구에서는 squid 처럼 캐시속의 하나의 object를 캐시 교체의 대상으로 삼는 것이 아니라 URL-net을 통해 얻어진 참조 패턴에 속한 object들의 그룹을 대상으로 한다. 각 그룹에 Storage LRU Expiration Age를 부여하고 같은 방식으로 운영함으로써 참조 패턴에 속한 그룹내의 object들이 한꺼번에 움직이도록 할 수 있다.

캐시 적중이 일어났을 경우에는, 캐시 서버에서 참조 패턴을 이용해 다음 요구될 가능성이 높은 URL페이지들을 미리 메모리에 로드하여 응답시간을 감소 시킬 수 있다.

7. 결론

인터넷 사용량의 폭발적인 증가로 인해 인터넷 정체현상이 큰 문제가 되고 있다. 이를 해결하는 방법으로 캐시 서버의 사용이 보편화 되어 있다. 그 중에서도 프락시 서버에 캐시를 설치하는 경우가 가장 활발히 연구 되고 있다. 이러한 캐시 서버의 성능을 향상 시키기 위해서는 보다 효율적인 캐싱 전략이 필요하다. 이를 위해서는 해당 네트웍 사용자들의 웹 사용 성향을 파악하는 트래픽 분석이 선행되어야 한다.

본 논문에서는 보다 입체적이고 효율적인 트래픽 분석을 위하여 URL-tree, URL-net이라는 자료구조를 제안하였다 이 두 자료구조를 이용한 트래픽 분석을 static, dynamic의 두 방법으로 수행하였다. Dynamic 분석의 경우에는 squid 서버의 수정을 통해 이루어졌다 이러한 분석을 통하여 얻은 정보는 병렬 캐시 서버 구축시 load 분산, 또는 캐시 교체에 이용될 수 있다.

참고문헌

[Alo92] R.Alonso and M Blaze, "Dynamic hierarchical caching for large-scale distributed file systems,"Proceedings of the 12th International Conference on Distributed Computer Systems, June 1992

[Bra94] P. De Bra and R Post, "Information Retrieval in the WWW: Making client-based searching feasible," http://www.win.tue.nl/win/cs1ist/reimpost/www94/www94.html

[Car95] Carlos R, Cunha, Azer Bestavros, Mark E Crovella "Characteristics of WWW Client-based Traces", July 1995

[Cha96] A.Chankunthod, P.B.Danzig C.Neerdaels, M.F Schwartz, and K.J.Worrell,"A hierarchical internet object cache," Proceedings of USENIX 1996, 1996

[Dan93] P B.Danzig, M.F Schwartz and R.S. Hall,"A case for caching file objects inside internet-works," ACM SIGCOMM 93 Conf , Sep 1993,

[Fis] G Fischer, http://www.tu-chemnitz.de/~ffgaam/httpd/scr/cache.html

[Gha] Ghaleb Abdulla, Edward A. Fox, Marc Abrams, Stephen Williams, "WWW Proxy Traffic Characterization with Application to Caching"

[Gla94] S Glassman, "A caching relay for the www," Proceedings of the 1st International WWW conf., 1994,

[Han] Hans-Werner Braun, Kimberly Claffy, "Web traffic characterization: an assessment of the impact of caching documents from NCSA's web server" Sep 1994

[ICM] ICM's Caching Survey -Results, http://wwcache.icm.edu.pl/survey/results

[Igo] Igor Tatarinov, "Performance Analysis of Cache Policies for Web Servers"

[Jun] J.Jung and K. Chon,"Nation-wide caching project in Korea - Design and Experiment " http://cosmos.kaist.ac.kr/~jyjung/Papers/proposition_paper.html

[Kim] K.Kim and K.Chon, "Update Policies for Network Caches " http://cosmos.kaist.ac.kr/salab/pu..echnicl-memo/SAL-TM-75/index.html,

[Kwa95] T T Kwan and R.E.McGrath, "NCSA's World Wide Web Server:Design and Performance " IEEE Computer Nov.,1995,

[Luo94] A Luotonen and K Altis, "World Wide Web Proxies," Proceedings of the 1st International World Wide Web Conference, 1994,

[Van94] G.Van Oosten, post to comp.infosystems.www, Feb,1994

[Ven] Venkata N. Padmanabhan, Jeffrey C. Mogul, "Using Predictive Prefetch to Improve World Wide Web Latency"

[Wes96] D.Wessels,"Internet Cache Protocol - history and future " ICM Workshop on Web Caching, Sep, 1996