접근 주기와 실제 갱신 주기에 기초한 리프레싱 기법

김인태, 김기창,
인하대학교

Refreshing technique based on reference period and update period

In-Thae Kim, Ki-Chang Kim
Dept. of Computer Science & Engineering INHA Univ.

요약 최근 들어 인터넷 데이터를 사용자와 가까운 위치 복사해 놓음으로써 인터넷 병목 현상을 줄이는 인터넷 캐시 서버의 사용이 증가되고 있다. 캐시 서버의 성능은 캐시 적중률과 캐시 된 데이터의 신선도에 의해 좌우된다고 할 수 있다. 데이터의 신선도는 캐싱된 데이터가 가장 최근에 갱신된 데이터와 일치할 확률이다. 신선도 유지를 위해 가장 많이 사용되고 있는 방법은 데이터마다 만료시간을 부여하여 만료시간이 지났을 때 새로운 데이터를 요청하는 방법이다.하지만 적절하지 않은 만료시간의 설정은 네트웍 교통량 증가나 사용자에게 신선하지 않은 데이터를 전달하는 문제가 생긴다. 본 논문에서는 각 데이터의 접근 주기와 실제 갱신주기에 기초하여 그 페이지의 리프레시 시간을 설정함으로써 요청될 가능성이 높은 페이지들만을 선택하여 프리패칭이 될 수 있도록 하는 기법을 제안한다.

1. 서론

인터넷 사용량이 급증함에 따라 인터넷의 정체현상이 큰 문제로 나타나고 있다. 이러한 문제를 해결하고자 많은 사람이 자주 접근하는 인터넷 페이지들을 미리 사용자 컴퓨터에 혹은 프락시 서버에 복사해 놓음으로써 웹 서버에 접속하지 않고도 데이터를 얻을 수 있도록 해주는 캐시 서버의 필요성이 대두되고 있다. 인터넷 교통량의 폭발적인 증가에 대해서는 많은 보고서가 그 현황을 보고하고 있다. 특히 HTTP 요청은 94년 인터넷 교통량의 19%를 차지하던 것이 96년에는 40%나 차지한다고 보고되어 있다. [1]는 NCSA 웹 서버에 대한 조회가 94년 2월부터 같은 해 12월까지 3-4개월마다 1,000,000 조회씩 증가한다는 것을 보고하고 있는데 현재 NCSA 서버는 평균하루에 3,000,000 조회가 이루어지고 있는 것으로 알려져 있다. 이러한 인터넷 교통량 분석을 토대로 인터넷 캐시에 대한 연구가 각국에서 활발히 진행되고 있다.

인터넷 캐시는 웹 서버측에 설치하는 경우와 사용자측에 설치하는 경우로 나눌 수 있다. 서버측에 설치하는 경우는 웹 서버의 페이지들을 미리 여러 캐시 서버에 복사하여 서버의 부하를 분산시키는 것으로서, 페이지마다 접근율을 정확히 알 수 있으므로 효율적인 캐싱이 가능하다. 특히, 접근율이 높고 자주 변경되는 페이지에 대해서는 변경될 때 마다 주요 고객의 지역 캐시에 복제 시켜줄 수 있다는 장점이 있다. 하지만 서버 측에 캐싱된 페이지들은 여전히 사용자들에 의해 인터넷을 통해 접근되어야 하므로 인터넷의 교통량을 줄일 수는 없고, 서버가 사용자의 지역 캐시에까지 직접 복제를 수행하는 경우에는 복제로 인한 인터넷 교통량의 증가를 초래한다는 문제점이 있다. 이러한 문제점 때문에 인터넷 캐시에 대한 연구는 주로 사용자측 캐시에 집중되어 있다. 가장 활발히 연구되고 있는 것은 프락시 서버에 캐시를 설치하는 방법에 대한 것이다. W3C의 httpd 3.0[2], Harvest Object Cache[3], squid[4]등이 주요 캐싱 프락시 서버이다. 특히, squid는 Harvest 캐시 서버로부터 발전되었으며 조사[5]에 의하면 현재 유럽의 캐시 사용자중 약 70%을 차지하고 있다. 그 외에 [6]가 Lagoon 캐시 서버를 제안하였으며, [7][8]는 각각 웹 서버를 캐싱 프락시 서버로 전환하는 방법 및 코드를 제시하고 있다. 캐시 서버의 성능은 캐시 적중률과 캐시 된 데이터의 신선도에 의해 좌우된다고 할 수 있다. 캐시 적중률은 사용자가 요청한 임의의 URL이 캐싱되어 있을 확률이다. 데이터의 신선도는 캐싱된 데이터가 가장 최근에 갱신된 데이터와 일치할 확률이다. 캐시 적중률이 아무리 높아도 데이터의 신선도가 낮다면 캐시 서버의 성능은 낮게 평가 될 것이다. 반대로 데이터의 신선도가 아무리 높아도 캐시 적중률이 낮다면 이 역시 캐시 서버의 성능을 저하시키게 될 것이다. 본 논문은 캐시된 데이터의 신선도를 높이는 방법에 관한 것이다. 데이터의 신선도를 완벽하게 보장하는 방법은 매번 요청된 URL마다 그 URL의 웹 서버에게 'If-Modified-Since'를 보내어 갱신 여부를 확인하는 것이다. 갱신이 안되었다면 그 URL의 신선도는 이미 보장된 것이고 갱신되었을 경우는 갱신된 데이터를 전달 받아 신선도를 보장해 줄 수 있다. 이 방법은 심한 네트웍 교통량을 유발 하므로 사실상 사용 불가능하다. 캐시된 데이터의 신선도를 높이는 그 밖의 모든 기법들은 네트웍 교통량을 심하게 유발하지 않으면서 사용자에게 높은 신선도의 데이터를 제공하려는 노력이라고 할 수 있다. 웹 캐시에서의 신선도 유지 문제는 분산 파일 시스템[9][10]과 유사하다. 웹 캐시 서버에서 현재 가장 많이 사용되고 있는 방법은 분산 파일 시스템에서 NFS 프로토콜과 유사한 방법으로 페이지마다 TTL을 부여하고 페이지에 대한 요청이 있을 때마다 해당 TTL을 확인하여, 유효기간이 남아 있으면 캐시 적중으로 취급하고 지났으면 캐시 미스로 간주하는 방법이다. 이 기법은 요청이 있다고 해서 무조건 'If-Modified-Since'를 보내어 갱신 여부를 확인하지 않고 자체적으로 해당 페이지에 대해 부여된 TTL값을 보고 TTL값이 만료되었을 때만 'If-Modified-Since'로 갱신 여부를 확인하기 때문에 지나친 네트웍 교통량을 피할 수 있다. 하지만 만료 되었을 경우 갱신된 데이터를 전송 받는 동안 사용자가 기다려야 한다는 문제점이 있고 갱신 시점이 주로 많은 사용자들이 네트웍을 이용하는 낮 시간일 확률이 높아 네트웍 교통량을 악화시킨다는 문제점이 있다. 이러한 문제점들을 해결하기 위해 TTL값이 만료된 URL들을 미리 사용자들의 접근이 줄어드는 밤시간 등에 갱신 시켜 놓는 방법을 생각할 수 있다. 이 경우는 요청될 가능성이 높은 페이지들만을 프리패치(prefetch)함으로써 시스템의 부하를 줄일 수 있는 방법이 강구되어야 한다. 본 논문은 프리패치 방식에 근거를 두고 각 페이지의 접근 주기와 실제 갱신주기에 기초하여 그 페이지의 리프레시(refresh) 시간을 설정함으로써 요청될 가능성이 높은 페이지들만을 선택하여 프리패칭이 될 수 있도록 하는 기법을 제안한다.

본 논문의 구성은 다음과 같다. 제 2장에서는 staleness문제 해결을 위한 기존 연구를 설명하고 제 3장에서는 문제 해결을 위한 새로운 접근방법과 구현을 설명한다. 제 4장에서는 결론을 짓는다.

2. Staleness 문제 해결에 대한 기존연구

그림 2.1은 staleness문제 해결을 위한 기존 방법들을 분류해 놓았다. 우선 누가 문제 해결을 주도하는 가에 따라 서버측, 캐시 서버측 두 분류로 나누어진다. 캐시 서버측 방법을 보면 staleness를 해결하는 '강','약' 정도에 따라 나누어 질 수 있는데 여기서 '강', '약'은 단지 데이터의 신선도 측면만 볼 때 쓰이는 의미이고 캐시 서버의 효용을 나타내는 그 밖의 요소(예:네트웍 교통량)가 가지는 의미가 아니다. 약한 대처방법은 staleness검사 시기에 따라 두 분류로 나누어진다.

2.1 Server Invalidation 기법

이 기법은 자신의 데이터가 어디에 캐시 되었는지 기억한 후 그 데이터가 변하면 데이터를 가지고 있는 캐시 서버에게 데이터가 더

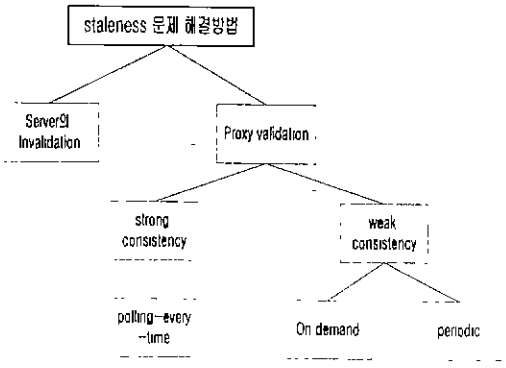


그림 2.1 staleness 문제 해결을 위한 방법 분류

이상 유효하지 않음을 알리고 이를 받은 캐시 서버는 유효하지 않은 데이터를 서버에 다시 요청하여 데이터가 유효하도록 하는 방법이다 [11] 이렇게 함으로써 캐시 데이터의 유효성을 강력히 유지할 수 있는 장점을 지니고는 있지만 서버는 더 이상 유효하지 않은 데이터가 현재 어디에 캐시 되었는지를 기억하고 있어야 되고 유효하지 않음을 알리기 위한 메시지를 보냄으로써 네트웍 교통량이 증가된다는 단점을 지니고 있다 또한 HTTP 프로토콜에 아직 데이터가 유효하지 않음을 알리는 메시지를 포함하고 있지 않아 기존 서버의 변화를 주어야 한다는 문제점이 있다. 비록 네트웍 교통량의 증가를 개선하는 방법이 Krishnamurthy에 의해 제안되어져 있지만 기존 서버들을 변화 시켜야만 한다는 문제점은 여전히 갖고 있다.[12]

2.2 Proxy validation 기법

HTTP/1.1[13]은 캐시된 데이터의 신선도를 높이기 위해 2가지 모델을 제공한다. 그 중 하나는 모든 URL마다 얼마나 변하지 않을 것인가를 나타내는 만료시간 필드를 가지게 함으로써 캐시 서버에 복사된 URL이 신선한지 판단하는데 사용할 수 있게 하는 것으로 HTTP/1.1에서는 이를 Expiration Model이라고 정의하고 있다 두 번째는 URL이 변했는가를 판단할 수 있도록 서버에 'If-Modified-Since' 필드를 붙여 요청하는 것으로 HTTP/ 1.1에서는 이를 Validation Model이라고 정의하고 있다 이 두 모델을 이용한 기법은 다음과 같다

- Polling-every-time 요청이 올 때마다 서버에 페이지의 적합성 판명을 요구하는 방법으로 HTTP 프로토콜에 정의 된 'If-Modified-Since' 헤더필드를 이용하여 쉽게 구현될 수 있다 항상 신선한 페이지를 받을 수 있다는 장점이 있지만 매번 다시 요청하므로 클라이언트에 대한 응답시간이 길어지고 네트웍 교통량이 증가된다는 단점을 가지고 있다.
- TTL (time-to-live) 데이터의 유효시간을 미리 정하고 유효시간이 지나면 새로운 데이터를 요청하는 방법으로 HTTP 프로토콜에 정의 된 'expires' 헤더필드를 이용하여 쉽게 구현될 수 있다

일반적으로 HTTP에서 정의하고 있는 두 모델을 혼합하여 TTL이 만료되었는가를 확인하여 만료되었다면 'If-Modified-Since'를 서버에 보내서 데이터의 유효성을 검사한다. 하지만 웹 페이지에 대한 정확한 만료시간을 제공하지 않으므로 현재시간에 일정한 상수를 더하는 정도의 산술로 TTL을 정하게 된다 이러한 방법은 적당한 TTL을 정하는데 상당히 어려운 점이 있다. 가령 그 값이 작다면 데이터가 변하지 않았더라도 자주 'If-Modified-Since'를 보내게 되고 크다면 적합하지 않은 데이터를 클라이언트가 볼 가능성이 높아지게 되는 문제점이 발생하므로 적당하게 TTL을 정하는 방법들이 연구되고 있다 [14]

만료시간의 체크 시기 관점에서 나누어 보면 요청이 올 때 체크와 주기적인 체크로 나눌 수 있다. 요청이 올 때 만료시간을 체크하는 방법은 클라이언트가 요청 때마다 매번 만료시간 체크를 기다려야 하고 만료시간이 지났다면 서버에게 유효성 검사를 요청하고 기다려야 하는 문제점이 있다 또 다른 방법인 주기적 만료시간 체크는 적당한 체크 주기를 설정하는 것이 중요한 점이라 할 수 있다.

실제로 많이 사용되는 캐시 서버인 squid의 경우 adaptive TTL [1] 을 사용한다. 그 알고리듬[16]은 아래와 같다

```
알고리듬 2.1 squid의 리프레싱 필요 여부 확인 알고리듬
입력 : 페이지 요청
출력 : 리프레싱 필요 여부
방법 :
if (CLIENT_MAX_AGE)
    if (OBJ_AGE > CLIENT_MAX_AGE)
        return STALE
if (OBJ_AGE <= CONF_MIN)
        return FRESH
if (EXPIRES) {
        if (EXPIRES <= NOW)
                return STALE
    else
                return FRESH
}
if (OBJ_AGE > CONF_MAX)
        return STALE
if (LM_FACTOR < CONF_PERCENT)
        return FRESH
return STALE
```

OBJ_DATE : 데이터를 서버로부터 받은 시간, HTTP의 Date 필드로 얻어낸다
OBJ_LASTMOD 마지막으로 변경된 시간, HTTP의 Last-Modified 필드로 얻어낸다.
OBJ_AGE 데이터를 가져온 후 경과 시간
OBJ_AGE = NOW - OBJ_DATE
LM_AGE: 데이터가 변경된 시간부터 데이터를 가져온 시간까지 경과된 시간
LM_AGE = OBJ_DATE - OBJ_LASTMOD
LM_FACTOR : LM_FACTOR = OBJ_AGE/LM_AGE
CLIENT_MAX_AGE: 클라이언트가 OBJ_AGE를 얼마나 허용하는지 정한시간, HTTP/1.1의 Cache-Control 필드로 얻어낸다.
EXPIRES : HTTP의 expire 필드로 얻어낸다.
CONF_MIN, CONF_PERCENT, CONF_MAX는 미리 설정해 놓는 값

출력이 STALE이면 요청된 페이지는 리프레시 되어야 하고 FRESH이면 현재 캐시된 페이지를 그대로 사용자에게 제공한다 squid가 요청된 페이지의 신선도를 판단하는 기준은 이 페이지의 서버가 EXPIRE 값을 정했을 경우와 그렇지 않을 경우에 따라 다르다 이러한 값들이 주어졌을 때는 서버가 정하는 유효기간이 지났는지 여부에 따라 신선도를 판단한다. 하지만 대부분의 페이지들은 이러한 유효기간에 대한 정보를 갖고 있지 못하다. 그런 경우는 시스템이 임의로 정한 값 CONF_MIN, CONF_MAX, CONF_PERCENT등과 해당 페이지의 체류시간 등을 참작하여 신선도를 결정한다.

3. 접근주기와 실제 갱신주기에 기반한 리프레싱 기법

3.1 리프레시를 위한 기본 접근방법

본 논문에서 제안하는 리프레싱 기법은 Proxy Validation기법에 속하며 만료시간의 체크를 주기적으로 하면서 만료시간을 좀 더 적당하게 정하는 새로운 접근을 시도하여 효율적인 리프레싱 전략을 세우고자 한다. 캐시된 데이터의 적당한 만료시간을 정하기 위한 우리의 접근방법은 각 클라이언트가 URL를 요청하는 주기에 초점을 맞추는 것으로, 각 URL별 클라이언트의 접근주기를 분석하여 다음에 접근할 시기를 미리 예측하고 클라이언트의 요청이 있기 전에 미리 리프레시 함으로써 staleness를 줄이고자 한다.URL의 클라이언트별 접근주기를 알아보기 위해 약 4 개월 간 인하대학교 한 세그먼트의 네트웍 사용현황을 조사하여 Static URL 리스트를 작성, 이를 이용하여 접근주기를 구하고 URL에 대한 접근주기가 가장 짧은 클라이언트를 선택하여 그 주기를 초기 주기로 정한다.URL요청 중 많은 수를 차지 하는 이미지 파일들은 오래도록 변하지 않는다는 특징을 감안하여 좀 더 긴 만료시간으로 정한다. 주기적으로 만료시간을 체크하여 만료시간이 지났다면, reference pattern[17]과 같은 Url group을 이용하여 프리패치 한다. HTML 파일을 프리패치 할 경우 HTML파일을 피싱(parsing)하여 포함되어 있는 이미지가 캐싱되어 있는지 확인하고 없다면 캐시되지 않은 이미지를 프리패치 한다.

3.2 알고리듬

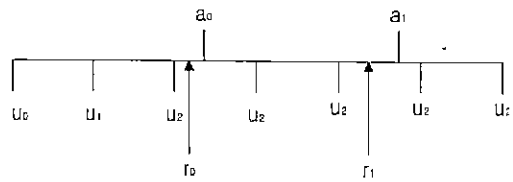
Static 분석에 의해 캐시된 모든 URL들에 대해 리프레시 시간이 설정되었다고 가정한다 리프레싱으로 인한 시스템 부하를 경감시키기 위해 각 URL들은 적당한 숫자의 리프레싱 시간군으로 분류된다. 각 리프레시 시간을 미리 알림이 맞추어져서 알람이 울리면 그 해당 시간군에 있는 URL들에 대한 리프레싱이 일어난다.

[1] 데이터의 생명시간에 초점을 맞추어 변화주기 설정

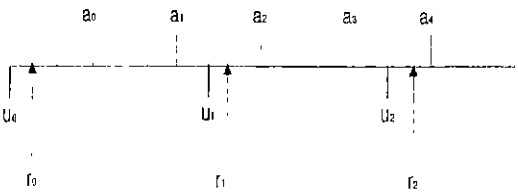
새로 캐시에 삽입되는 URL의 경우 디폴트 리프레시 시간군은 하루 주기 시간군이다. 알람은 사용자들의 접근이 격감되는 반시간으로 주로 조정되어 있다 알람이 울렸을 때 리프레싱 알고리듬은 다음과 같다

알고리듬 3.1
입력 리프레싱 시간군에 속하는 URL들
출력 그 URL들의 리프레싱
방법 :
현 시간군의 모든 URL에 대해
1. clientHttpRequest 구조체 생성
2 If-Modified-Since를 보낸다
3 변경이 안되어 있으면
접근주기를 고려하여 그 URL 시간군 조정
변경이면
페이지 패치, 새로 추가된 어미지 파일 요청
제한을 가지고 이 URL의 시간군 감소

알고리듬은 기본적으로 현 리프레시 시간군에 속하는 모든 URL들의 신선도를 검사한다 아직 변경되지 않은 URL들은 그 URL에 대한 접근주기를 고려하여 다음 시간군으로 이동시킨다. 오랫동안 갱신되지 않거나 접근되지 않는 URL들은 주기가 큰 리프레시 시간군으로 일려 나가도록 하기 위해서 이다 이미 변경된 것으로 확인되면 해당 페이지를 패치한 후 'Last-Modified' 필드를 조사한다. 캐시 서버의 예측보다 일찍 변경되는 경향이 파악되면 역시 리프레시 시간을 조절해야 한다. 리프레시 시간군의 적절한 조정을 위해 각 URL마다 접근시점, 실제 갱신 시점, 리프레시 시점들을 계속 추적해야 한다. 접근 시점(a_i)는 그 URL이 접근될 때 마다 기록되면 그 URL이 최근에 어떠한 주기로 접근되고 있는가를 보여준다. 실제 갱신 시점(u_i)은 그 URL이 실제로 갱신될 때 마다 'Last-Modified' 필드를 통해 얻어지며 그 URL이 최근에 어떠한 주기로 실제 갱신되고 있는가를 보여준다 그 URL의 최적 리프레시 시점(r_i)은 u_i와 a_i에 의해 결정된다. 그림 3.1은 리프레시 시점을 결정하는 두 경우를 나타내고 있다. ui의 주기가 ai보다 빠른 경우에는 ai에 맞춰져야 하며 늦은 경우에는 ui에 맞춰져야 한다. 어느 경우든 r_i는 u_k가 일어난후 그 다음 첫번째 a_j이 일어나기 전에 수행되어야 한다.



a)갱신주기가 접근주기보다 빠른경우



b)접근주기가 갱신주기보다 빠른 경우

그림 3.1 접근주기와 갱신주기를 기반한 리프레시 주기 설정

3.3 구현

리프레시 알고리듬은 squid의 소스를 변경하여 수행되었다. 변경된 부분을 설명하기 위해 squid의 리프레시 코드부분을 간략히 소개한다. 우선 클라이언트가 URL을 요청하면 요청된 URL이 이미 캐시된 URL인지 확인한다. squid는 캐시된 URL마다 'StoreEntry'라는 구조체를 생성하고 있어서 이미 캐시된 URL인가 확인하기 위해 해당 URL의 StoreEntry 존재여부를 확인한다. StoreEntry는 다음과 같이 정의 되어있다

```
struct _StoreEntry {
        const cache_key *key;
        MemObject *mem_obj;
        .
        .
        .
};
typedef struct _StoreEntry StoreEntry;
```

만약 이미 캐시된 URL이라면 리프레시 알고리듬에 의해 리프레시를 해야 하는지 확인 후 리프레시를 해야 한다면 'If-Modified-Since'를 이용해서 neighbor 또는 서버에 요청한다 클라이언트가 squid 캐시서버에 URL을 요청했을 때 생성되는 구조체를 살펴 보면 요청한 URL이 예전에 이미 캐시되어 있는 상태라고 가정하고 클라이언트로부터 요청이 오면 'clientHttpRequest' 라는 구조체를 만들고 entry 포인터 필드로 URL을 통해 찾은 StoreEntry를 가리키도록 한다 그리고 알고리듬1에 의해 리프레시를 해야할지 결정한다. 리프레시를 해야 한다면 새로운 StoreEntry를 생성하고 요청 한 URL이 변했을 경우 새로 생성한 StoreEntry에 저장한다 clientHttpRequest는 다음과 같이 정의 되어있다

```
struct _clientHttpRequest {
        request_t *request;
        char *uri;
        StoreEntry *entry;
        . .
};
typedef struct _clientHttpRequest clientHttpRequest;
```

static URL 리스트로부터 URL별 각 클라이언트의 접근주기를 계산하여 가장 짧은 접근 주기를 선택하여 refresh 주기파일 생성하고 캐시서버에 캐시되지 않은 URL의 요청이 오면 static refresh주기 파일에서 요청된 URL을 검색 한다 URL이 있다면 dynamic refresh 주기 파일에 URL, Expiration time, 주기를 저장하고 없다면 주기를 최소 단위로 정한 후 저장하고 정기 리프레시 시간이 되면 알고리듬3.1로 리프레시 한다

4. 결론

인터넷 정체현상을 해결하고자 캐시 서버의 사용이 증가되고 있다 이러한 캐시 서버 성능은 캐시 적중율과 캐시된 데이터의 신선도에 좌우된다 본 논문에서는 캐시된 데이터의 신선도를 유지하는 기법으로써 접근주기와 갱신주기를 기초로 리프레시 주기를 정하였다 이를 프리패치에 이용하여 보다 높은 신선도 유지 기법을 제안하고 구현하였다

참고문헌

[1] T T.Kwan and R E Mcgrath, "NCSA's World Wide Web Server·Design and Performace," IEEE computer Nov.,1995
[2] A Luotonen and K Altis, "World Wide Web Proxies," Proceedings of the 1st International World Wide Web Conference, 1994
[3] A.Chankunthod, P.B.Danzig, C Neerdaels, M F Schwartz, and K.J.Worrell,"A hierarchical intenet object cache," Proceedings of USENIX 1996, 1996
[4] D.Wessels,"Internet Cache Protocol - history and future," ICM Workshop on Web Caching, Sep 1996.
[5] ICMJ's Caching Survey Results,
http://w3cache.icm.edu pl/survey/results
[6] P. De Bra and R Post, "Information Retrieval in the WWW' Making client-based searching feasible,"
http'//www.win tue nl/win/cllist/reimpost/www94/www94 html
[7] G.Fischer, http //www tu-chemnitz.de/~flpaam/httpd/scr/cache html
[8] G Van Oosten, post to comp.infosystems www, Feb,1994
[9] Michael N. Nelson, Brent B. Welch, and John K. Ousterhout Caching in the Sprite file system. ACM Transactions on Computer Systems, pages6(1) 134-154, February 1988.
[10] John H Howard, Michael Kazar, Sherri G. Menees, David A Nichols, M. Satyanarayannan, Robert n. Sidebotham, and Michael J. West Scale and performance in a distributed file system. ACM Transactions on Computer Systems, pages 6(1)-51-81, February 1988
[11] C.Liu and P. Cao, Maintaining strong cache consistency in the World-Wide Web, Proceedings of the 17th IEEE International Conference on Distributed Computing Systems, May 1997
[12] Nalachander Krishnamurthy and Craig E. Wills Piggyback server invalidation for proxy cache cohrence
[13] Internet Engineering Task Force, Hypertext transport protocol-HTTP/1.1, January 1997
[11] H Kim and K.Chon, Update Policies for Network Caches
[15] V. Cate, Alex-a global file system, In Proceedings of the 1992 USENIX File System Wrokshop, pages 1-12, May 1992
[16] http://squid nlanr net/Squid/FAQ/FAQ-12 html#ss12 20
[17] Jeffrey C. Mogul, Hinted caching in the web. In Proceedings of the 1996 SIGOPS European Workshop, 1996