Load Balancing in Distributed Web Server Systems With Partial Document Replication

Ling Zhuo, Cho-Li Wang and Francis C. M. Lau

Department of Computer Science and Information Systems

The University of Hong Kong

Presented by: Cho-Li Wang

Outline

- Introduction
  - Extensible Distributed Web Server (EDWS)
- Document Distribution in DWS
- Three Algorithms
  - Greedy-cost
  - Greedy-load/cost
  - Greedy-penalty
- Performance Evaluation
- Conclusion & Future Work

The Challenges

- 1996: Netscape Web site (November):
  - 120M hits per day
- 1998: Olympic Winter Games (Japan):
  - 634.7M (16 days), peak day 57M.
- 1999: Winbledon,
  - 942 M hits (14 days), peak day 125M, (> 7K hits/sec)
- 2000: Olympic Games 2000 :
  - peak day 502.6 M, peak 10K/s

The Challenges

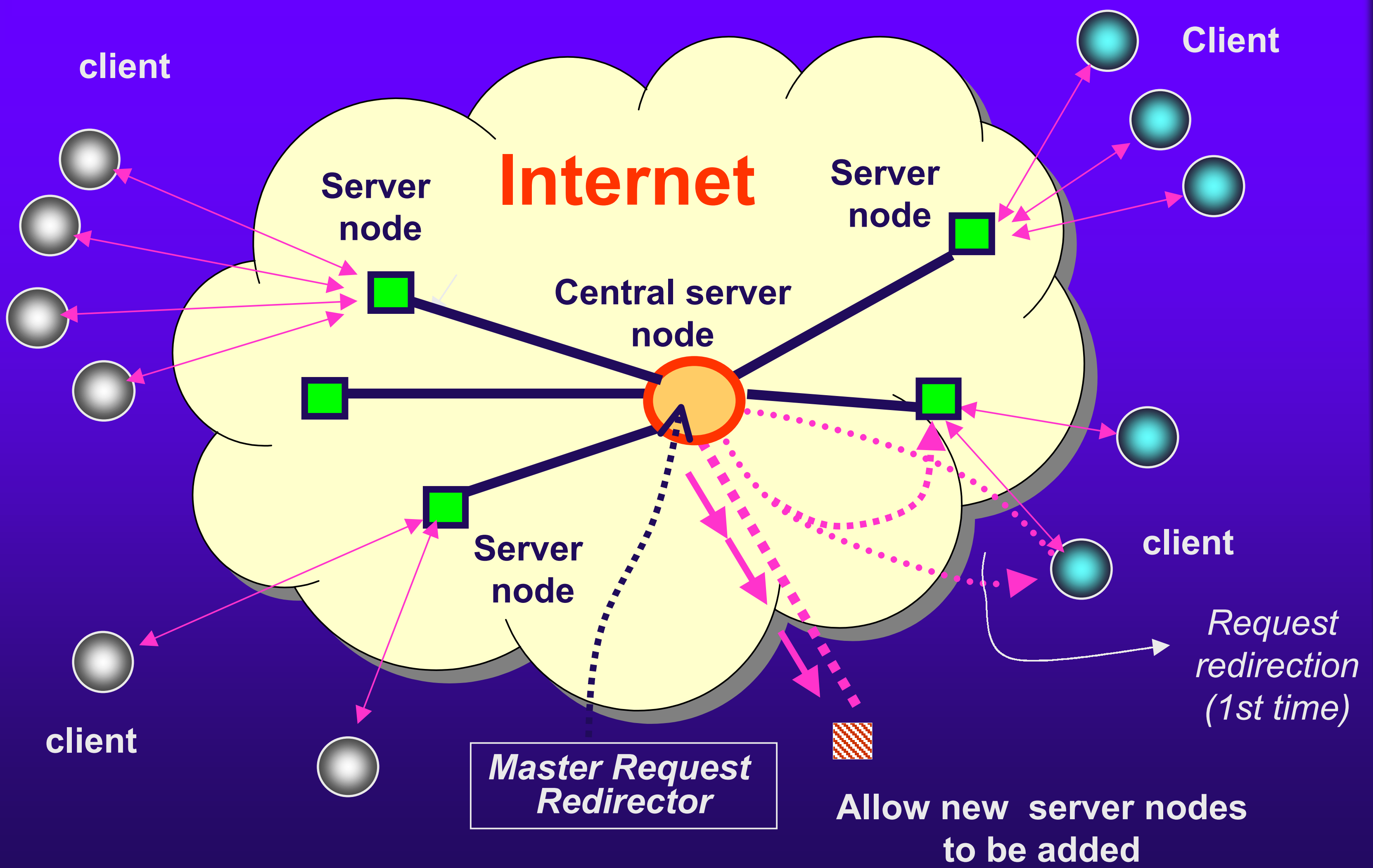
- **More people are getting online**
  - How many online: 407 million in November 2000 to 544 million in February 2002.
  - More broadband users: 57% of the workers in U.S access Internet via broadband in office. The figure will be more than 90% by 2005. Home broadband user will also increase from less than 9M 2001 to over 55M by 2005 [IDG report]

The increasing popularity of the World Wide Web has resulted in large bandwidth demands which translate into high latencies (response time) perceived by Web users.

Ways To Reduce Response Time

- Web Proxy Caching
  - Web Proxy (e.g., Squid)
- More Powerful Web Server
  - A monolithic Web Server
    - advance hardware support (E.g., SMP, faster backbone network) and optimized server software (E.g., JAWS, Flash,…)
  - A Cluster Web Server :
    - With high-speed load balancing switch (Layer 7/4 dispatching), Cooperative Caching,..
    - E.g., SWEB, LARD, LVS+Apache, and HKU’s p-Jigsaw and Cyclone.

Extensible Distributed Web Server (EDWS)



Extensible Distributed Web Server

- Main Features of EDWS
  - Traffic/Load is distributed over multiple server nodes
  - Allow servers to be added or removed.
  - No full mirroring of Web site documents
  - Using standard HTTP Redirection protocol for routing the Web requests
  - Periodically replicate and re-distribute documentations among servers based on access record of last period and the current configuration to achieve load balancing.

Document Distribution Scheme

- **Document distribution scheme:**
  - Rules that determine how documents are replicated and placed in a DWS
- **Performance Issues**
  - Load balancing
  - Communication cost of document redistribution

Existing Schemes

- **Full replication : NCSA server**
  - Waste of storage resources
  - DNS-based dispatching : Partial control on incoming requests
- **Non replication : DCWS, SWEB**
  - Content-aware routing : Bottleneck in the central dispatcher
  - Load balancing through Document Migration; can not deal with “hot” documents.
- **Partial replication :**
  - Content-based routing
  - Load balancing through statically or dynamically replication and redistribution of documents based on current global load status

Existing Partial-replication Schemes

- **Dynamic Approaches**
  - Documents are dynamically replicated based on current global load status
  - E.g., DC-Apache (Univ. of Arizona), P-Jigsaw Parallel Web Server (HKU), WhizzTech’s WhizzBee.
- **Static Approaches**
  - Documents are replicated and placed statically based on past access pattern
  - E.g., RobustWeb
- **Disadvantage**
  - Cannot achieve good load balancing
  - Traffic caused by updating the document replication and distribution is rarely discussed

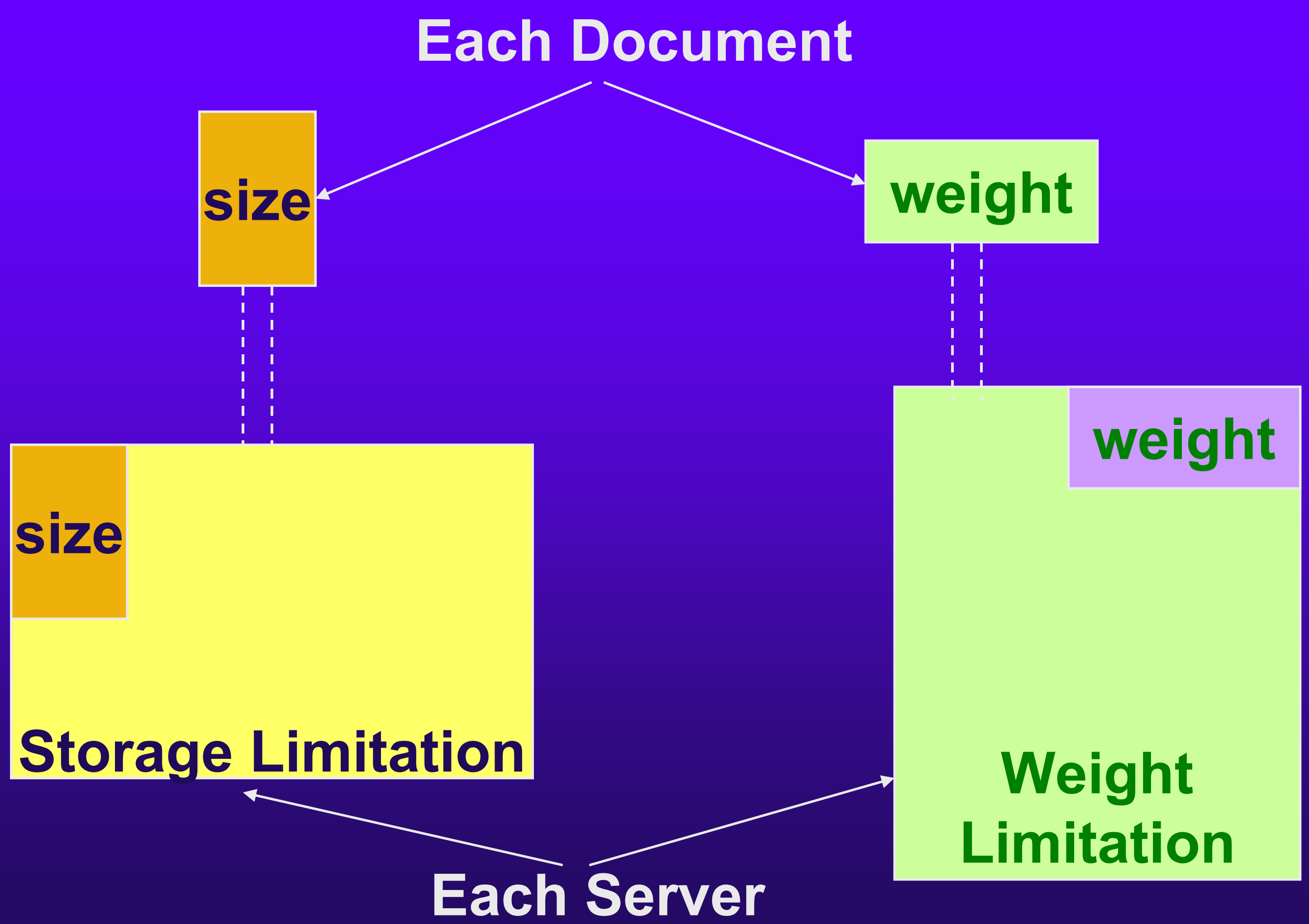
Overview of Document Distribution Scheme in EDWS

- Main Steps :
  - Analyzing the access log files, and computing the weight of each document

    w = access rate in the last period * size

    - representing the predicted workload a document to bring to the EDWS
  - Apply the ***density algorithm*** to compute the replica number of each document with the consideration of disk space limit
  - Distributing the documents and their replicas to the server nodes

Storage Limit vs. Load Balancing



Density Algorithm

- A document's "density" represents the predicted workload per unit storage of a document brings to a server (You can view it as "popularity").

  d = *w* / size of the document

- <u>Number of replicas proportional to density</u>
  - Duplicate more copies for frequently requested documents ("hot pages") -- More effective for load balancing
- <u>Maximize storage utilization:</u>
  - Replicating as many documents as the storage capacity allows

Density Algorithm

Input: d_i, s_i, C, M, N, **Output:** c_i $(i = 1, \ldots N)$

Variables:
- S, total size of document
- S_disk, available disk space;
- d_{min}, minimal density
- $temp\_S$, total size of temporary replicas
- $temp\_c_i$, temporary number of replicas

Main Steps:

1. compute S, $S\_disk = M * C - S$
2. sort documents by decreasing density d_i, and find d_{min}
3. **for** $i = 1$ to N $\{temp\_c_i = d_i / d_{min}\}$
   compute $temp\_S$
4. **for** $i = 1$ to N $\{$
   $c_i = temp\_c_i * S\_disk / temp\_S$ */* scaling */*
   if $(c_i >= M\text{-}1)\{$
   $c_i = M\text{-}1$, $temp\_S = temp\_S - temp\_c_i * s_i$
   $S\_disk = S\_disk - c_i * s_i\ \}\}$
5. finally decide c_i $(i = 1, \ldots N)$ $/* ++c_i */$

Distributing the Replicas

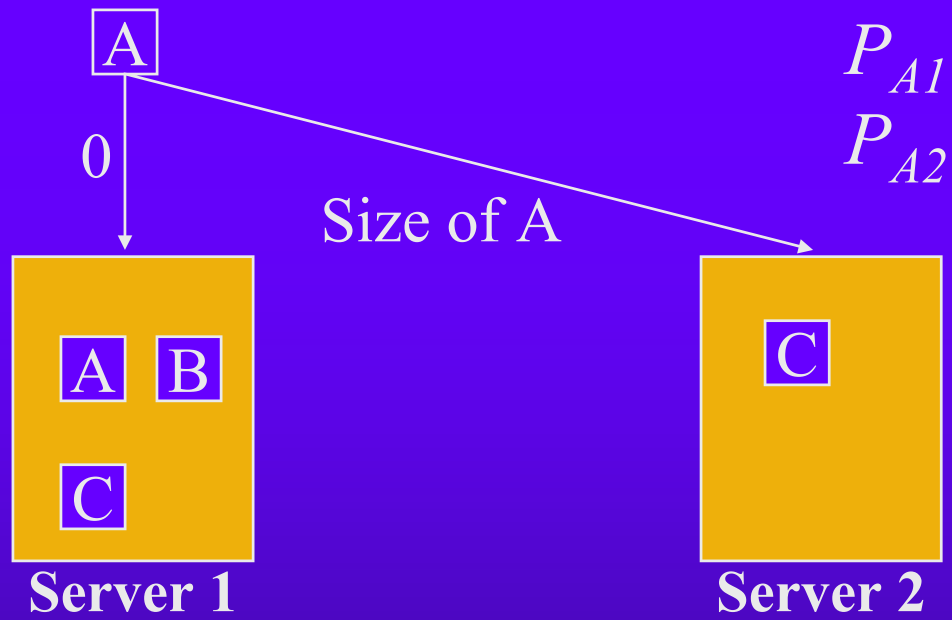
- **Main goals**
  - Balancing the load among the server nodes
  - Minimizing document redistribution traffic
- **Method:**
  - A "cost link" is constructed between each document and each server
  - cost link (redistribution cost) =
    - 0 (if local) or
    - the size of the document (if remote)
- **Optimization Problem:**
  - NP-hard, see a brief proof in the paper

Problem Formulation

- *N* documents, ***M*** servers
- Each document has size of s_i and number of replicas c_i , $i = 1, \ldots N$.
- "cost link" p_{ij} : the number of bytes to be transferred if document *i* is assigned to server *j*; for $i = 1, \ldots N$ and $j = 1, \ldots M$
- Replica assignment: $t_{ij}^{\,l}$ $(l = 1, \ldots c_i)$,
  - 1 if *l*th replica of *i*th document is placed on *j*th server; otherwise 0.
- The determination of c_i is under the limitation of total storage, i.e., .

$$\sum_{i=1}^{N} (s_i * c_i) \leq M * C$$

Cost Link : An Example



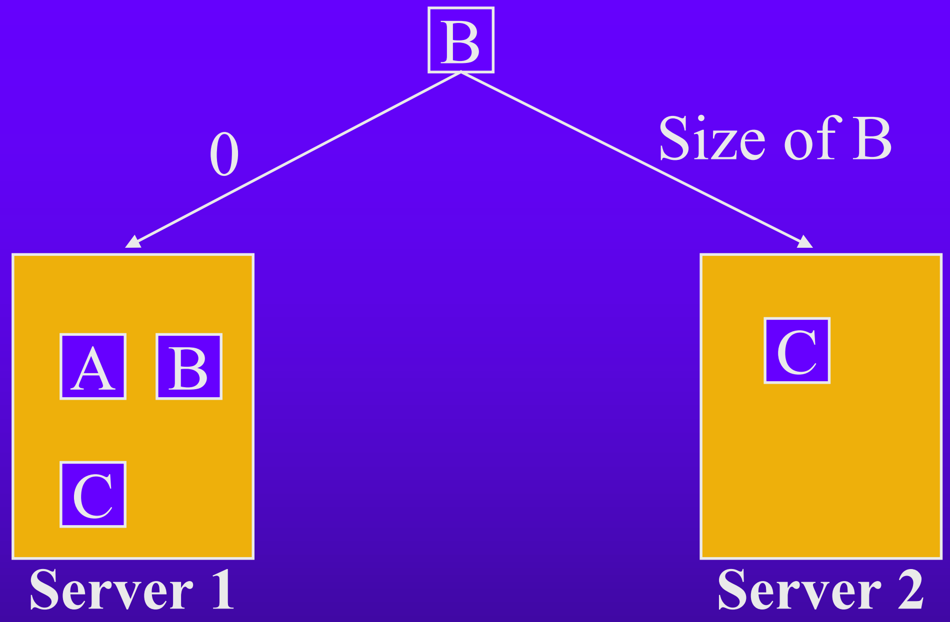
$P_{A1} = 0$

$P_{A2} = \textit{size of } A$

N documents, ***M*** servers. Each document has size of s_i and number of replicas c_i , $i = 1,\ldots N$.

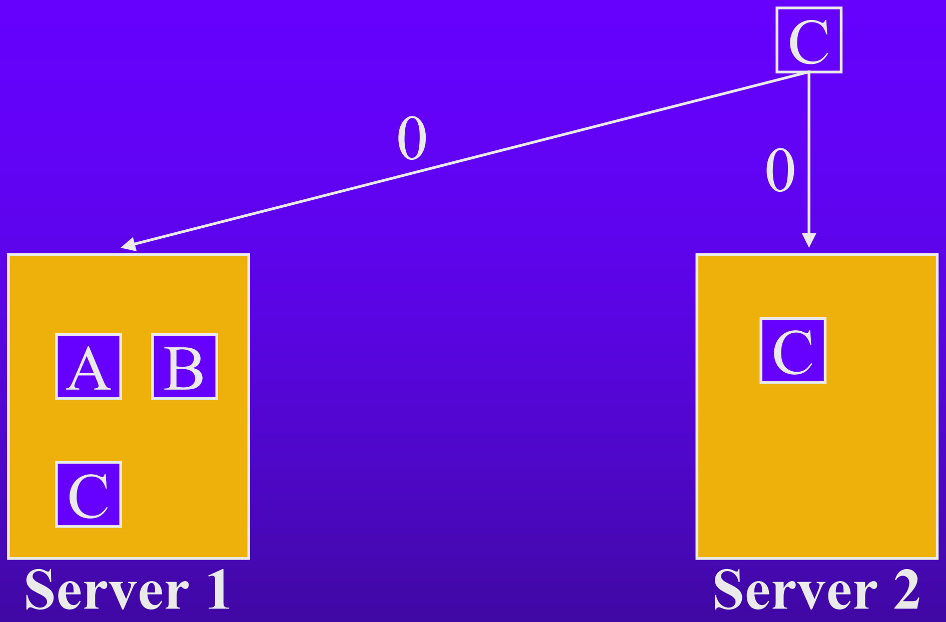
"cost link" p_{ij} : the number of bytes to be transferred if document i is assigned to server j; for $i = 1,\ldots N$ and $j = 1,\ldots M$

Cost Link



$$P_{B1} = 0$$
$$P_{B2} = size\ of\ B$$

Cost Link



$$P_{C1} = 0$$
$$P_{C2} = 0$$

Algorithm 1 : Greedy-cost (GC)

- Basic idea:
  - Minimizing redistribution cost by keeping as many documents as where they are located
  - No consideration of load balancing
  - No guarantee hot pages are fully duplicated
- How ?
  - Sort the pairs (document, server node) by the value of "cost link" (p_{ij}) between them, increasingly, and distribute the documents in this order
- Possible Disadvantages:
  - Cannot adapt to the change of access pattern quickly

Algorithm 1 : Greedy-cost (GC)

Input: c_i, s_i, p_{ij}, C, M, N

Output: $t_{ij}^{\,l}$ ($i = 1,\ldots N$, $j = 1,\ldots M$, $l = 1,\ldots c_i$)

1.sort (i, j) pairs by increasing cost, p_{ij}

2.**for** each (i, j) in the sorted list{

if ($c_i > 0$) {

allocate a replica to server j if it has

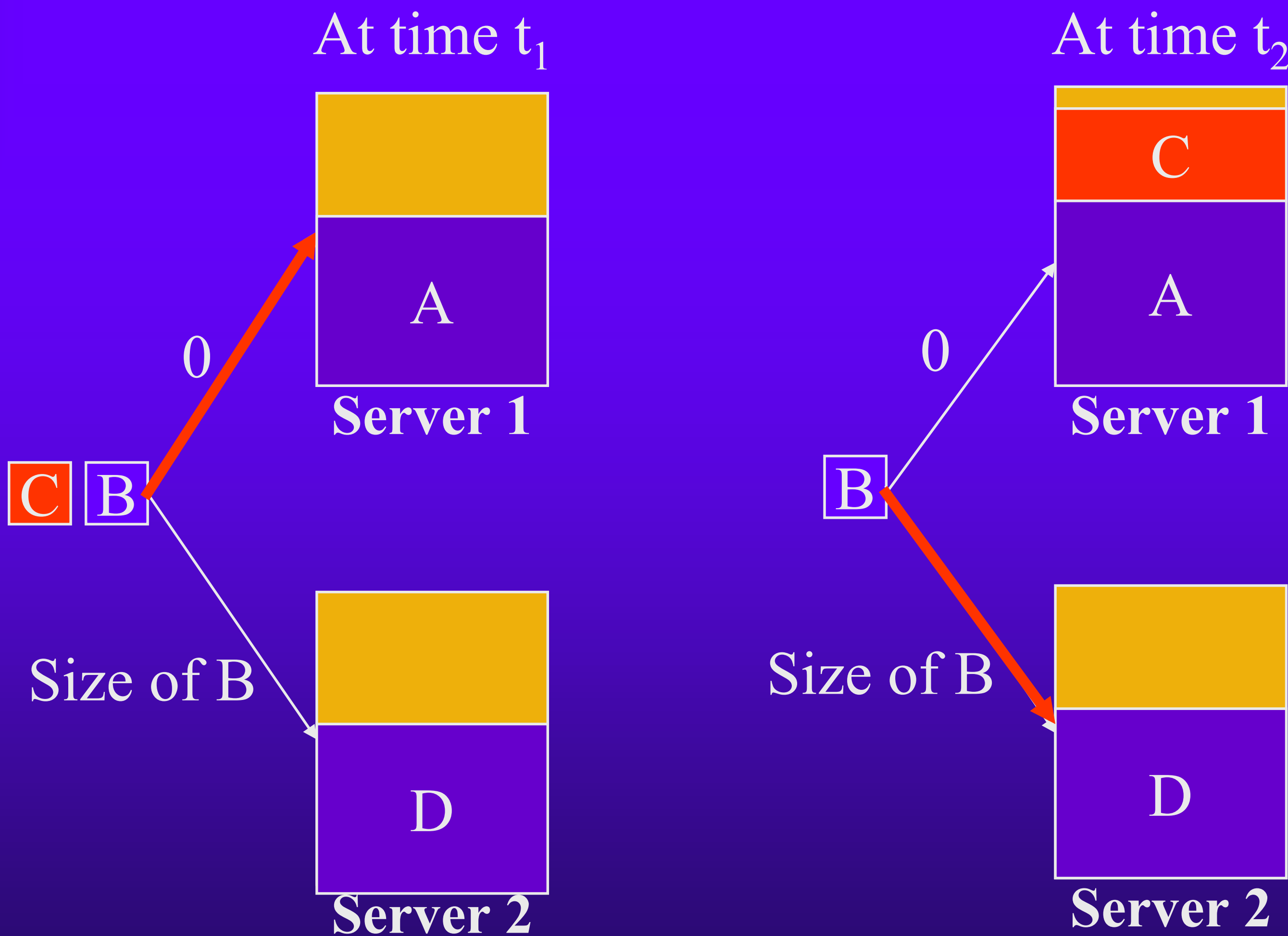
enough space and $t_{ij}^{\,l} = 0$ ($l = 1,\ldots c_i$).

$c_i = c_i - 1$ }}

Algorithm 2 : Greedy-load/cost

- **Basic idea:**
  - Mainly consider the load balancing
  - Enforce popular Web pages being fully duplicated
  - Also consider the redistribution cost
- **How ?**
  - Sort the **documents** by their **densities** decreasingly and distribute the documents in this order -- process popular web pages first .
  - For each document *i*, sort the **cost link** p_{ij} increasingly, and select the top c_i **servers** in this order.
  - If same **cost link value**, select the server assigned with least workload at that time (enhance load balancing).
- **Possible Disadvantages:**
  - May not effectively reduce redistribution cost based on the above process order as it proposes.

“Penalty” due to different processing order :



Delay distributing B until time t2, server 1 may already be almost full. ***Penalty = size of B – 0***

Algorithm 3 : Greedy-penalty

- Basic idea:
  - Reduce the total traffic by determining a certain documents distribution order -- *General Assignment Problem*
- How ?
  - Sort the documents by their densities decreasingly
  - At each loop, for each remaining replica set *i*, we compute penalty, f_i as the difference in the costs of its best and second best placements that incurs less communication cost.
  - Select and process the replica set with least penalty (favor smaller page) and distribute it and its replicas.
- Disadvantage:
  - More computation needed: each loop we need to find the document with least penalty.

Algorithm 3 : Greedy-penalty

Input: $c_i, p_{ij}, s_i, C, M, N$

Output: $t_{ij}^{\,l}$ $(i = 1,\ldots N, j = 1,\ldots M, l = 1,\ldots c_i)$

Variables: f_j, penalty for document i $(i = 1,\ldots N)$

while there are unassigned replica sets {

for each unassigned replica set i{

if only c_i server nodes have enough storage to hold document i{ allocate replica set i

goto while /* completed */}

else { sort servers by increasing cost with document i, p_{ij}.

compute f_i }}

Sort replica sets in decreasing penalty, f_i

Allocate the replica set with minimal f_i in its best placement}

Time Complexity

- **Greedy-cost** $\Theta(MN\log MN + MN)$
- **Greedy-load/cost** $\Theta(NM \log M)$
- **Greedy-penalty** $\Theta(N^2 \log N + NM \log M)$

Experiment Setup

- Use the **CSIM** 18 package
- Homogeneous server nodes
- Disk seek time : **19 ms**
- Disk transfer rate : **21 MB/s**
- Initially, Web documents are randomly placed on the server nodes without replication.
- Documents distribution activated every 3 hours.

Dynamic Scheme

- For comparison, we simulate the DC-Apache (DC):
  - Periodically (every 10 minutes), check global load status
  - Replicate documents from overloaded server (load is 50% higher than average load)
  - Revoke documents from under-loaded server (load is lower than average load)

Metrics

- Load Balancing Metric (LBM):
  - Record the peak-to-mean ratio of server utilization every sampling period (10 minutes)
  - Smaller LBM → better load balancing
- Average total traffic per period

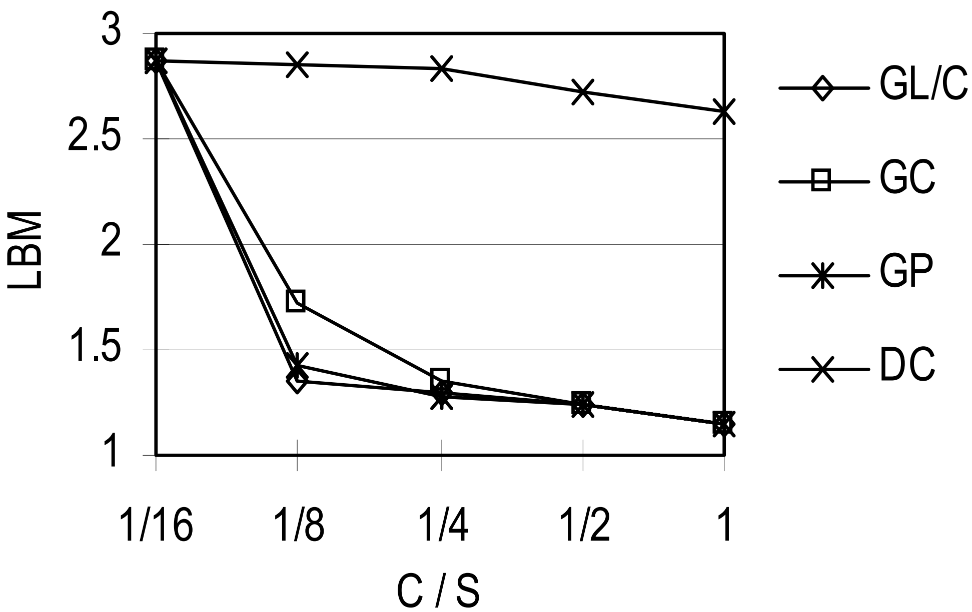
Data Sets

- Two real traces of Web access
  - Data Set 1 : a website used for hosting personal home pages,
  - Data Set 2 : The Internet Traffic Archive.
- Documents in the same directory are grouped and these groups are used as basic units of replication and distribution
- Duration of dataset: one day

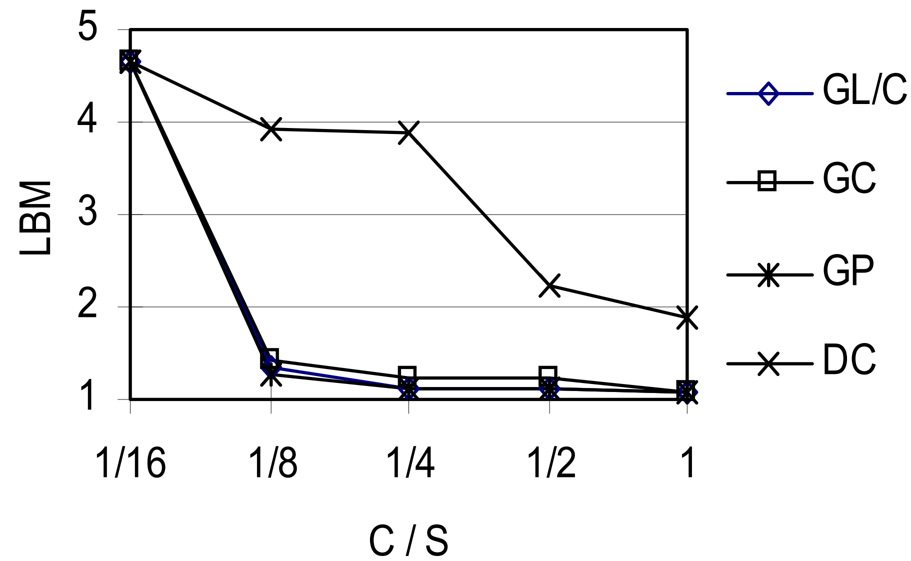
Load Balancing vs. Disk Capacity

C : the storage capacity of each server node

S : the total size of the documents



Data Set 1 (16 server nodes)



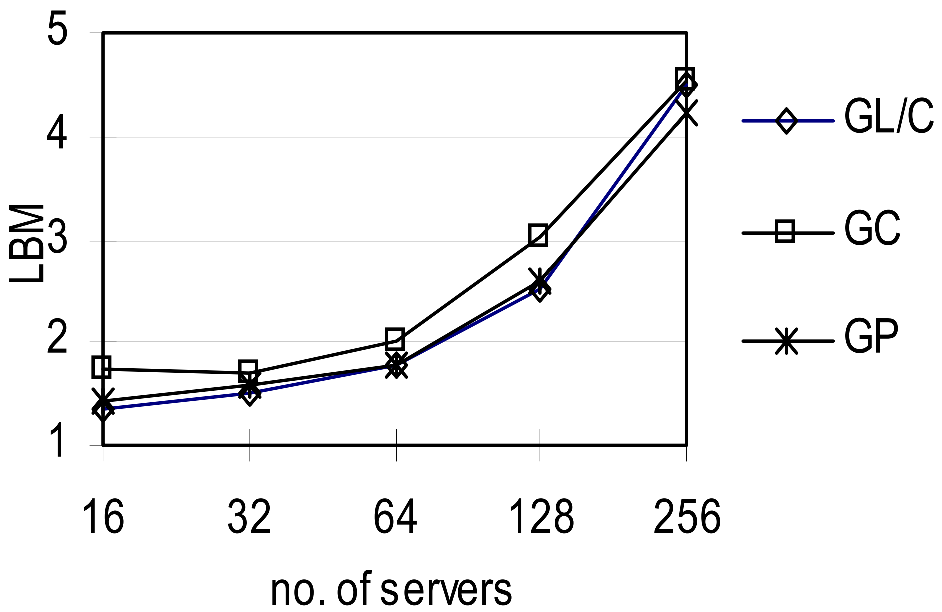
Data Set 2 (16 server nodes)

GL/C and GP are better than GC. DS is the worst -- doesn't fully utilize the available disk space.

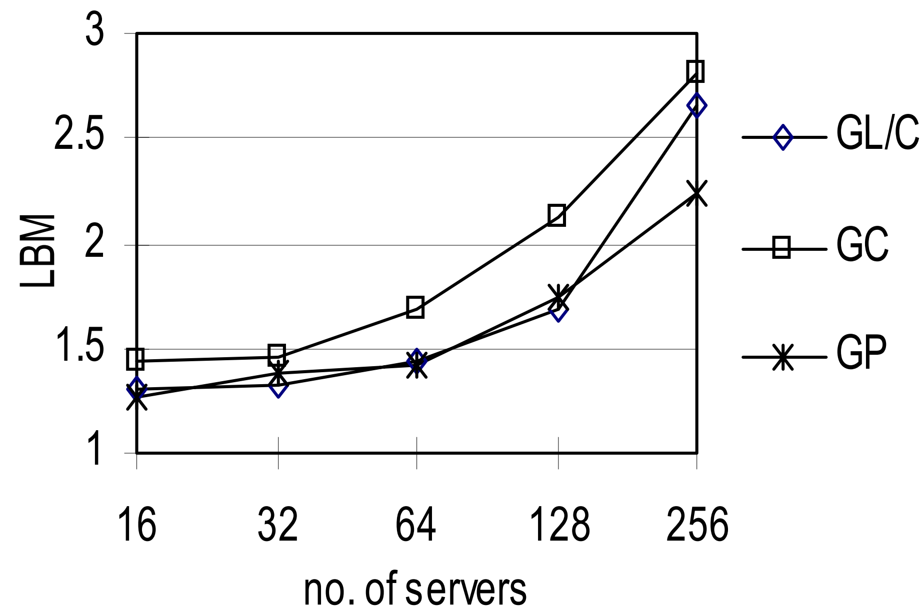
Load Balancing vs. No. of Servers

Fixed storage capacity (C = 1/8 S)

Scale the no. of servers : M= 16 ~256



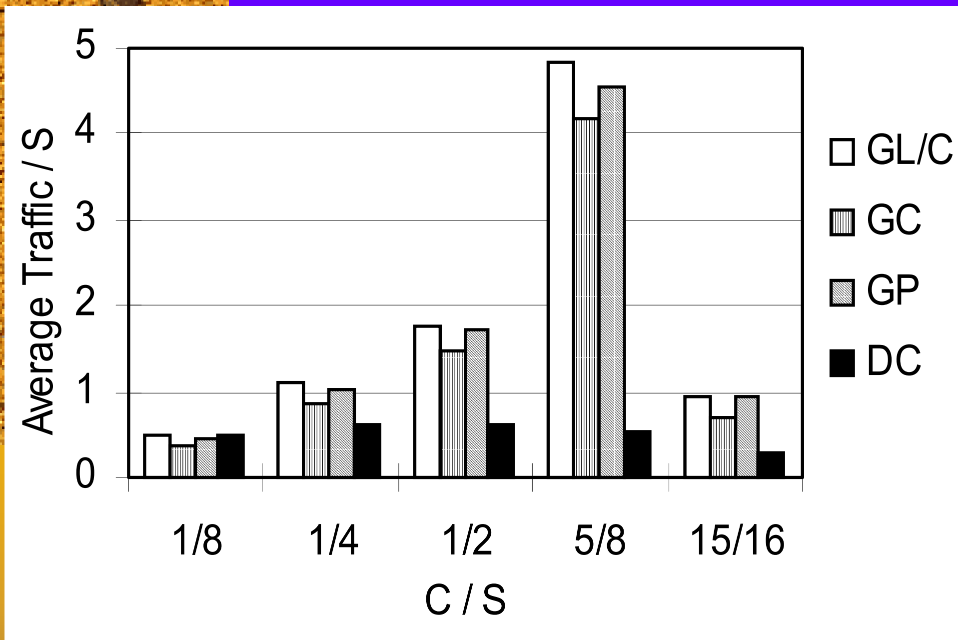
Data Set 1 (*C* / *S* = 1/8)



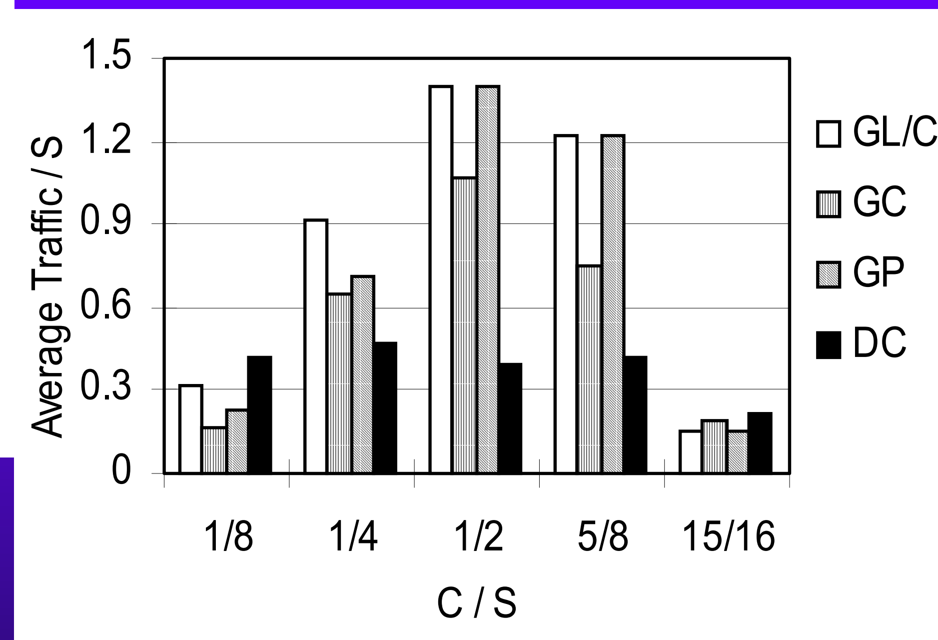
Data Set 2 (*C* / *S* = 1/8)

GL/C and GP are still close when the node number is not very large. When more than 128 nodes, GL/C appears to deteriorate faster than GP.

Average Traffic vs. Disk Capacity



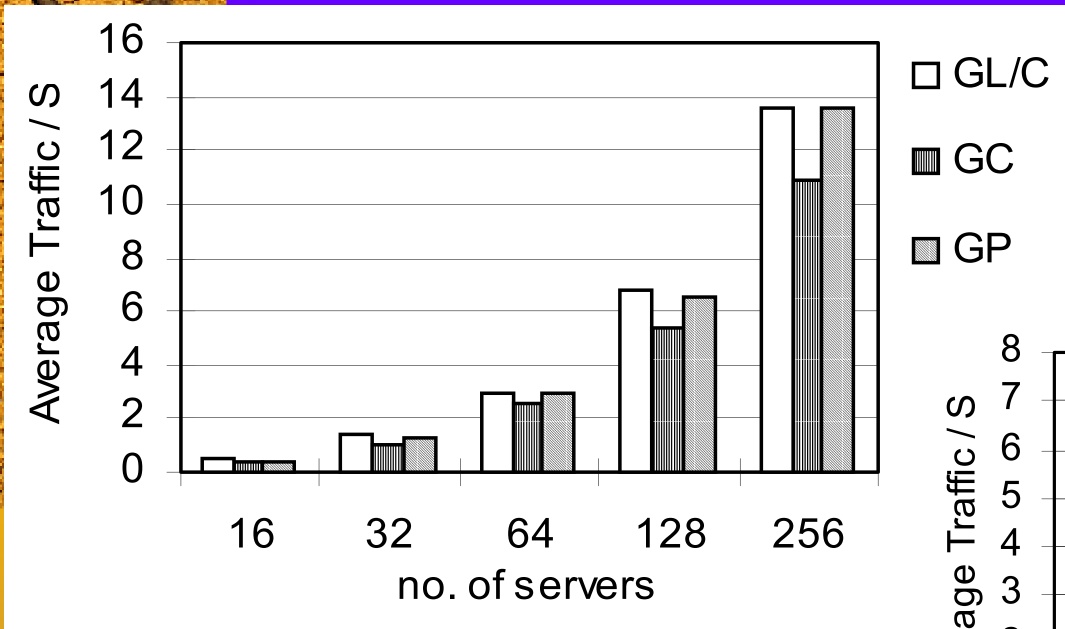
Data Set 1 (16 nodes)



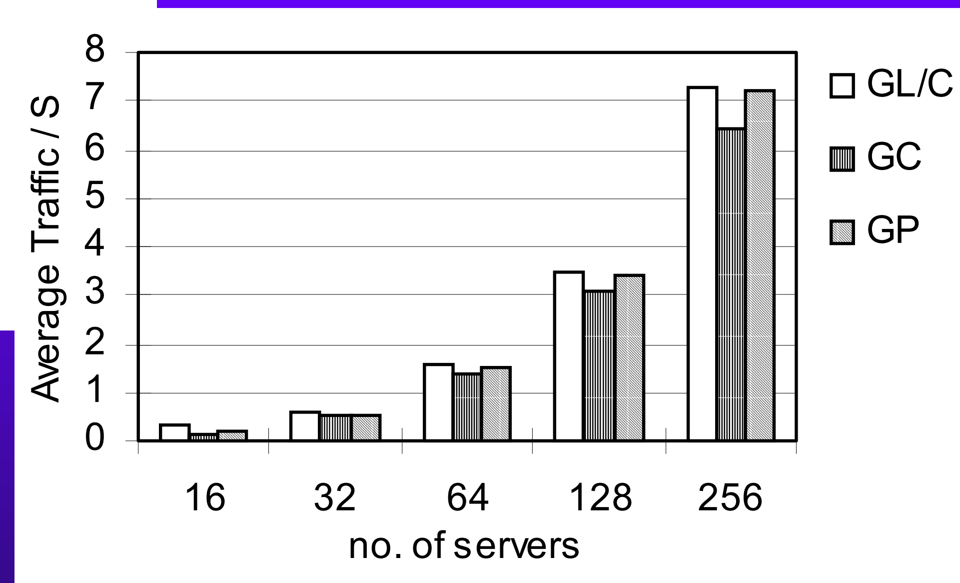
Data Set 2 (16 nodes)

- GC incurs the least cost.
- GP is better than GL/C, but when the storage capacity is large, the traffic caused by GL/C and GP is almost the same.

Average Traffic vs. No. of Servers



Data Set 1 (*C* / *S* = 1/8)



Data Set 2 (*C* / *S* = 1/8)

GC still causes least traffic, and the traffic caused by GL/C and GP get closer as the number of nodes increases.

Conclusions

- Compared with the dynamic scheme, our document distribution scheme can
  - Achieve better load balancing
  - Generate less internal traffic
  - Provide better Web service

Conclusions

- Greedy-cost
  - Generally, worst load balancing and least internal traffic
  - Easiest to be affected by initial placement of documents
- Greedy-load/cost
  - Generally, best load balancing
  - More traffic than Greedy-penalty
  - Least computation

Conclusions

- Greedy-penalty
  - Most stable load balancing performance
  - Most computation
- A suitable algorithm can be chosen according to the practical situation of a EDWS system

Future Work

- An on-line algorithm
  - Achieve similar load balancing
  - Further reduce internal traffic
- Proximity-aware algorithm
  - Achieve both network proximity and load balancing
- Document distribution scheme for **heterogeneous** EDWS systems