LARD
(Locality Aware Request Distribution)

By
Saumitra M Das

PURDUE UNIVERSITY

Outline

- Motivation
- Introduction & Background
- Basic and Detailed Idea
- Issues and Optimizations
- Performance Evaluation

PURDUE UNIVERSITY

Motivation

JUSTIFYING DOING ALL THIS WORK

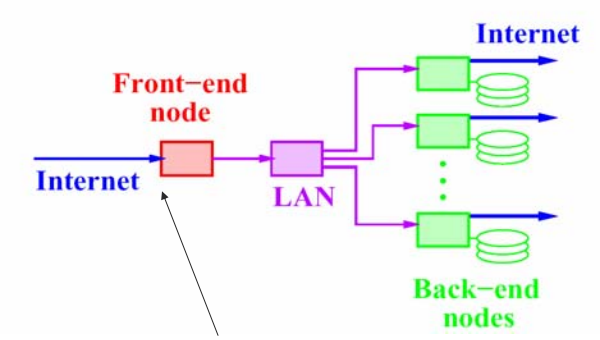
- Demand is growing on network servers
  - Why?, you ask..
    - Internet is growing
    - BW is increasing
    - Site consolidation
- Cheaper $olution$ are attractive
  - Inexpensive PCs and LANs
  - Parallelizable nature of job

PURDUE UNIVERSITY

3

Motivation

FOCUS OF THE WORK



How should this box behave so that we save money.

In other words, what policy should it use for request distribution so that throughput is maximized with minimum utilization of resources.

PURDUE UNIVERSITY

4

Motivation
DISSECTING THE TITLE

- Locality
  - Taking advantage of files already cached in back-end server's memory
  - What does this mean?
    - Requests accessing same data be sent to the same set of servers
- So, locality aware is…
- Is that all we need?
  - If so, you could go home instead of listening to me.

PURDUE UNIVERSITY

Motivation
DISSECTING THE TITLE

- It is only half the picture
- What about distribution ?
  - Load balanced system
    - Distribute requests evenly among back-end servers
- What do we want in the ideal case?
  - Improve hit-rate and response time
- Many studies focus only on one aspect and ignore the other

PURDUE UNIVERSITY

Introduction

REQUEST DISTRIBUTION PROTOCOL V 1.0

- Round Robin Request Distribution

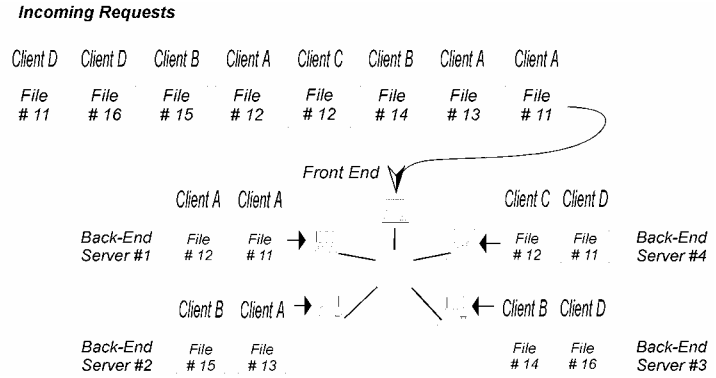
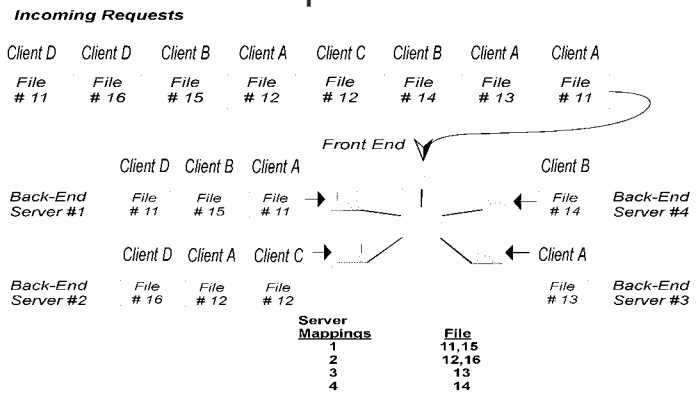


Diagram Source: A Khaleel, A.Reddy, TAMU

Introduction

REQUEST DISTRIBUTION PROTOCOL V 2.0

- File Based Request Distribution

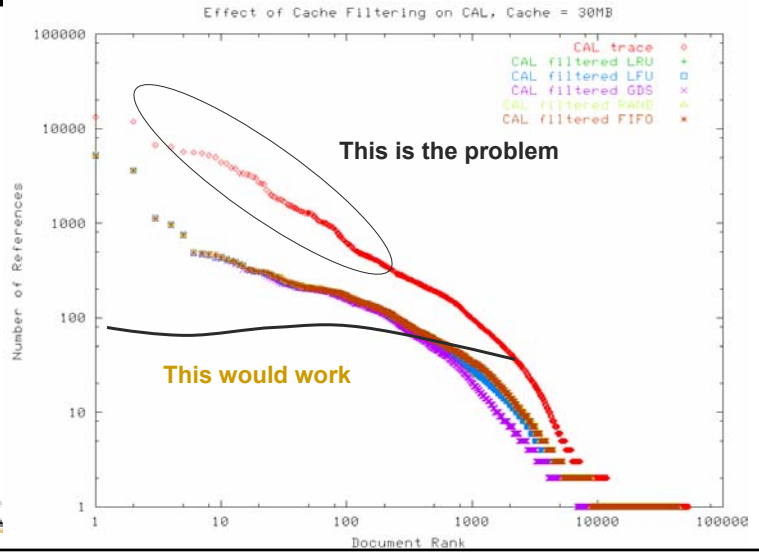


Will this work for a web workload ?

Diagram Source: A Khaleel, A.Reddy, TAMU

Introduction

IS REQUEST DISTRIBUTION PROTOCOL V 2.0 ALL WE NEED?



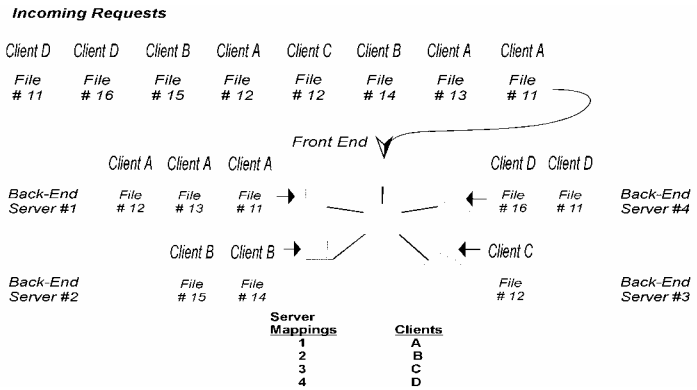
PUR UNIVE

9

Introduction

REQUEST DISTRIBUTION PROTOCOL V 3.0

- Client Based Request Distribution



PURDUE UNIVERSITY

10

Diagram Source: A Khaleel, A.Reddy, TAMU

Introduction
REQUEST DISTRIBUTION PROTOCOL V 4.0

- Locality Aware Request Distribution
  - Broadly based on file-based scheme
  - Addresses the issue of load balancing
  - **Each file assigned a dynamic set of servers instead of just one server**
  - **Each server assigned on average a static set of files instead of a dynamic set**

PURDUE UNIVERSITY

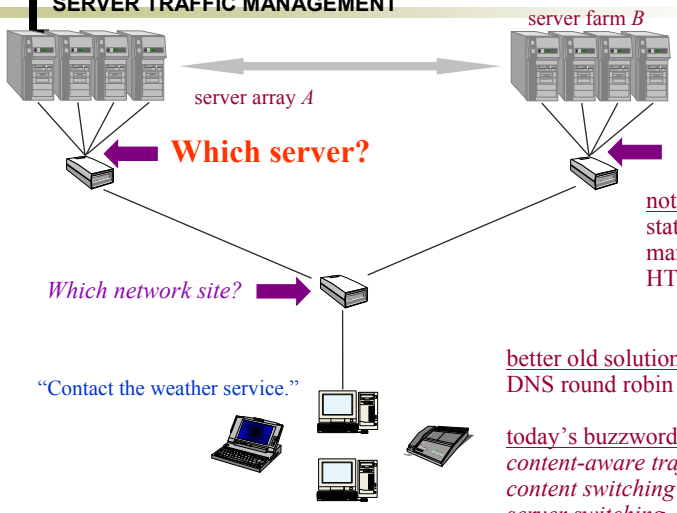
Introduction
CONTRIBUTION OF THE PAPER

- Scheme designed to achieve both locality and load balancing
- Simulation studies and comparisons
- Prototype Implementation and Evaluation

PURDUE UNIVERSITY

Background

SERVER TRAFFIC MANAGEMENT



not-so-great solutions
static client binding
manual selection
HTTP forwarding

better old solutions
DNS round robin [Brisco, RFC 1794]

today's buzzwords
content-aware traffic management
content switching (L4-L7)
server switching

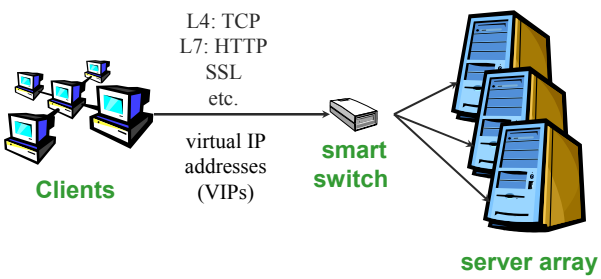
PURDUE UNIVERSITY

13

Source: Jeffrey Chase, Duke

Background

SERVER TRAFFIC MANAGEMENT



Goals
server load balancing
failure detection
access control filtering
priorities/QoS
external VIP management
request locality
caching

What to switch/filter on?
L3 source IP and/or VIP
L4 (TCP) ports etc.
L7 URLs and/or cookies
L7 SSL session IDs

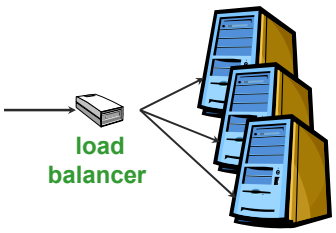
PURDUE UNIVERSITY

14

Source: Jeffrey Chase, Duke

Background

WHAT CAN YOU DO WITH A L4 SWITCH



Issues
switch redundancy
mechanics of L4 switching
handling return traffic
server failure detection (health checks)

Policies
random
weighted round robin (WRR)
lightest load
least connections

Key point: the heavy lifting of server selection happens only on connect request (SYN).

Performance metric: connections per second.

Limitations
connection-grained
no request locality
no session locality

PURDUE UNIVERSITY

15

Source: Jeffrey Chase, Duke

Background

HOW TO SELECT THE SERVER? – THE OLD WAY

```
/* initially index = 0,    L = (S0,S1,...,Sn-1) */
While (1) {
  i == index;
  if (i == 0) {
    cw--;
    if (cw <= 0) cw = max of W(Sn);
    if (cw == 0) return NULL;
  }
  if (W(Si) >= cw) {
    index = (i + 1) % n;
    return Si;
  }
  else
    index = (i+1) % n;
}
```

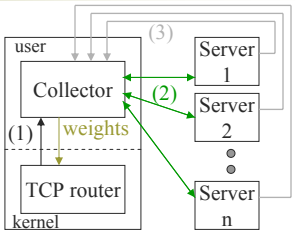
PURDUE UNIVERSITY

16

Background

L4 SWITCH + WRR

(1) number of active connections
(2) periodic polling (http request/response)
(3) periodic script execution and feedback
- queue length (cpu, disk)
- amount of allocated network buffer

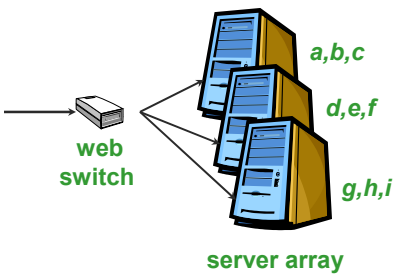


- Pros:
  - high routing throughput
- Cons:
  - Content-blind scheduling
  - Difficult to compute weights accurately
    - insufficient load information, load information feedback delay & overhead
  - Dynamic load imbalance across servers

PURDUE UNIVERSITY

Background

WHAT CAN YOU DO WITH A L7 SWITCH?



Idea: switch parses the HTTP request, retrieves the request URL, and uses the URL to guide server selection.

Example: Foundry
host name
URL prefix
URL suffix
Substring pattern
URL hashing

Issues
HTTP parsing cost
URL length
HTTP 1.1
session locality

Advantages
separate static content from dynamic
reduce content duplication
improve server cache performance
cascade switches for more complex policies

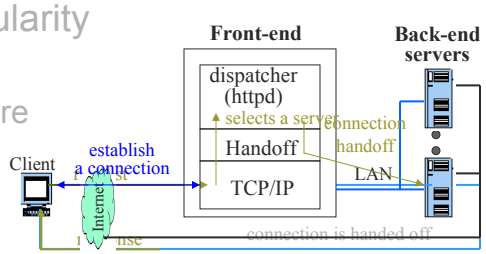
PURDUE UNIVERSITY

Source: Jeffrey Chase, Duke

Background

L7 SWITCH SCHEMATIC

- Scheduling granularity = connection
  - but, content-aware scheduling

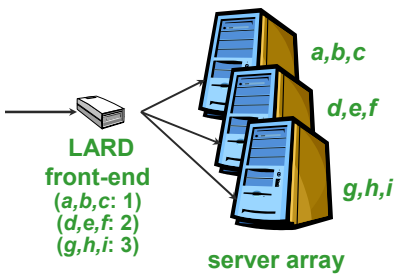


- Benefits
  - Performance improvement of back-end servers
    - cache affinity(locality) based scheduling
  - Easy to specialize the back-ends for certain types of content or services

PURDUE UNIVERSITY

Basic Idea

THE LARD ALGORITHM



LARD front-end maintains an LRU cache of request targets and their locations, and table of active connections for each server.

Idea: route requests based on request URL, to maximize locality at back-end servers.

Policies

1. *LB* (locality-based) is URL hashing.

2. *LARD* is locality-aware & LB: route to target's site if there is one and it is not "overloaded", else pick a new site for the target.

PURDUE UNIVERSITY

Source: Jeffrey Chase, Duke

Basic Idea

THE LARD ALGORITHM

```
while (true)
  fetch next request r;
  if server[r.target] = null then
    n, server[r.target] ← {least loaded node};
  else
    n ← server[r.target];
    if (n.load > T_high && ∃ node with load < T_low) ||
        n.load ≥ 2 * T_high then
      n, server[r.target] ← {least loaded node};
  send r to n;
```

PURDUE UNIVERSITY

Basic Idea

THE LARD ALGORITHM – WHAT IS LOAD?

- What is the load here
  - Load = # of active connections
- Why
  - Good summary metric
  - Robust
  - Available in front end
  - Simple
  - Quickly adapts

PURDUE UNIVERSITY

Basic Idea

THE LARD ALGORITHM - INTUITION

- T_{low} = load below which a back-end is likely to have idle resources
- T_{high} = load above which node is likely to cause substantial delay in serving a request
- How to choose these two?
  - The study chose 25 and 65

PURDUE UNIVERSITY

Basic Idea

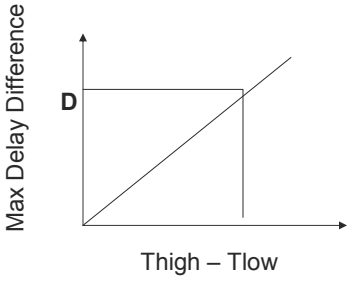
THE LARD ALGORITHM – HOW TO CHOOSE THESE

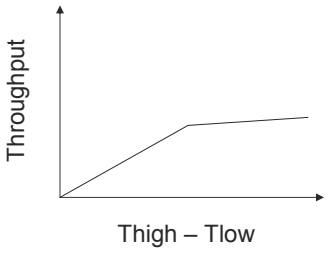
- T_{low} = f (back-end speed)
  - Choose T_{low} high enough to avoid idling of back-ends
- T_{high} is a different issue
  - $T_{high} – T_{low}$ should be low enough to limit the delay variance among the back-ends
  - $T_{high} – T_{low}$ should be high enough to tolerate short term load fluctuations without affecting locality

PURDUE UNIVERSITY

Basic Idea

THE LARD ALGORITHM – HOW TO CHOOSE THESE





$$T_{high} = (T_{low} + D/R) / 2$$

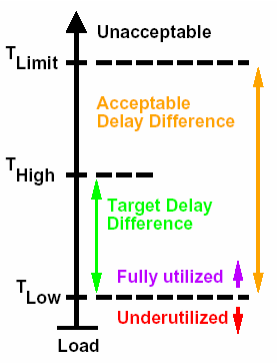
R = average request service time

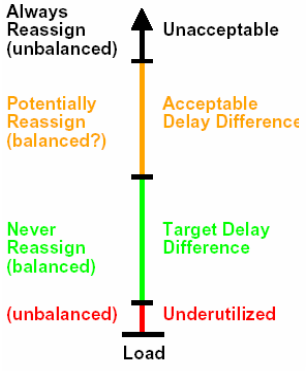
PURDUE UNIVERSITY

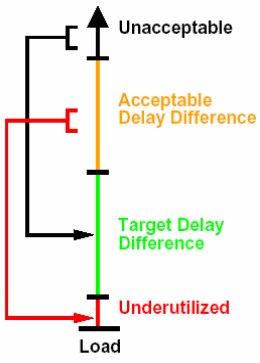
25

Basic Idea

THE LARD ALGORITHM – TO REASSIGN OR NOT TO REASSIGN THAT IS THE QUESTION







PURDUE UNIVERSITY

26

Issues

HOW & WHAT TO IMPROVE?

- WHAT
  - **If back-end serving target x fails ?**
    - **Replicate (in the paper)**
  - **If load on all nodes rises to 2*T_{high}?**
    - **Admission Control (in the paper)**
  - If the front-end is overwhelmed?
    - Distributed Distributor (follow up paper)
  - If the workload is primarily dynamic content?
    - LARD for Dynamic Content (attempt by students)
  - **If acting as a proxy is too much overhead for the front-end?**
    - **TCP Handoff not Splicing (in the paper)**
  - If connections are persistent
    - P-HTTP support [ARON99] (follow up paper)

PURDUE UNIVERSITY

Optimizations

REPLICATION

- Why
  - to prevent a single target to cause a back-end from overloading
- How
  - Any ideas…?

PURDUE UNIVERSITY

Optimizations

REPLICATION

- The front end maintains a mapping from targets to a set of nodes that serve the target.
- Requests for a target are assigned to the least loaded node in the target's server set.
- If a load imbalance occurs, the front end checks if the server set has recently changed or not(within k seconds), if so, it picks a lightly loaded node and adds that to the set.
- If a request has multiple servers and has not moved or had a new one added, the front end needs to remove one to keep the degree not too high.

```
while (true)
  fetch next request r;
  if serverSet[r.target] = ∅ then
    n, serverSet[r.target] ← {least loaded node};
  else
    n ← {least loaded node in serverSet[r.target]};
    m ← {most loaded node in serverSet[r.target]};
    if (n.load > T_high && ∃ node with load < T_low) ||
        n.load ≥ 2T_high then
      p ← {least loaded node};
      add p to serverSet[r.target];
      n ← p;
    if |serverSet[r.target]| > 1 &&
        time() - serverSet[r.target].lastMod > K then
      remove m from serverSet[r.target];
  send r to n
  if serverSet[r.target] changed in this iteration then
    serverSet[r.target].lastMod ← time();
```

PURDUE UNIVERSITY

Optimizations

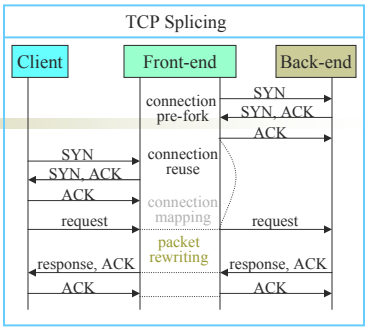
ADMISSION CONTROL

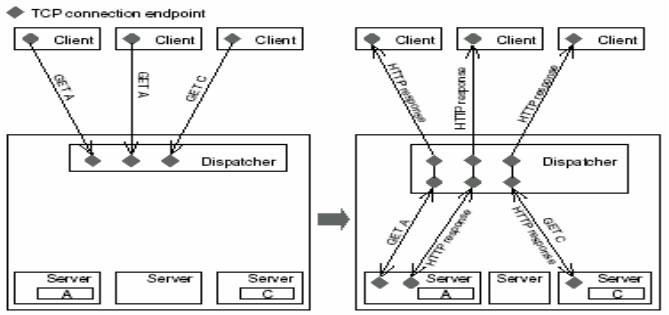
- Why
  - To prevent load on all nodes to become > 2*T_{high} and LARD=WRR
- How
  - Front-end limits total number of connections to
    - S = (n-1)*T_{high} + T_{low} - 1 , (n = number of back-end nodes)
  - At most (n-2) nodes can have a load >= T_{high} while no node has load < T_{low}
  - All n nodes can have load > T_{low}

PURDUE UNIVERSITY

Optimizations

TCP SPLICING





A. Clients set up connection with the dispatcher and send HTTP request to it.

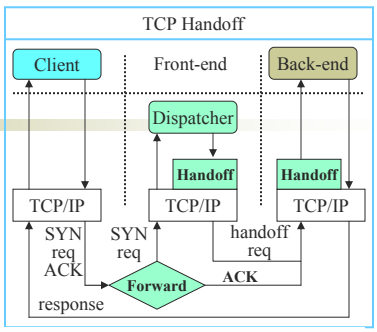
B. After parsing a request, the dispatcher sends it to the chosen web server node. Response from the web server is forwarded to the client by the dispatcher.

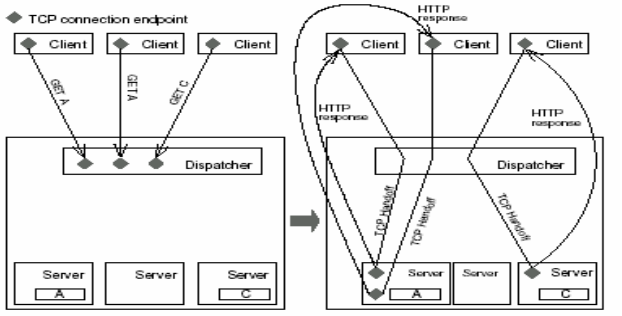
PURDUE UNIVERSITY

31

Optimizations

TCP HANDOFF





A. Clients set up connection with the dispatcher and send HTTP request to it.

B. After parsing a request, the dispatcher handoffs the connection to the chosen web server node. Response is sent directly to the client by the web server.

PURDUE UNIVERSITY

32

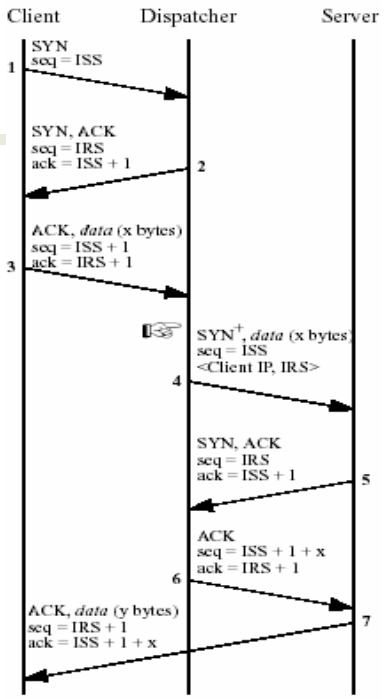
Optimizations

TCP HANDOFF – STATE TRANSFER

Key : individual server node can masquerade as the dispatcher as long as it has the appropriate connection state.

Caveats:

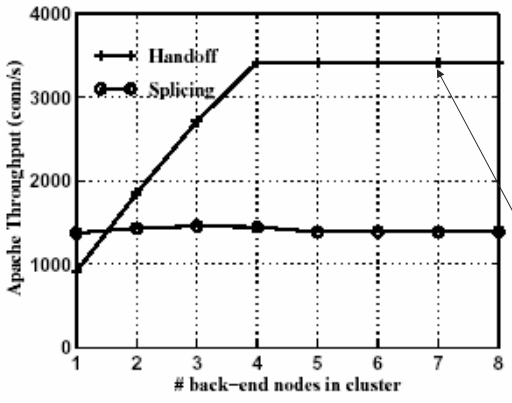
1. Dispatcher cannot accept any options such as SACK from the client which the backend does not support
2. Dispatcher must choose an initial flow control window that each server can satisfy



PURDUE UNIVERSITY

Optimizations

WHY IS HANDOFF BETTER



1. Splicing though better than a relay has the overhead due to HTTP response traffic
2. Handoff requires kernel modification to both frontend and backend but does not require HTTP response traffic to go through distributor

With handoff, the throughput scales with the cluster size more than splicing

PURDUE UNIVERSITY

34

Simulation Results

SIMULATION MODEL

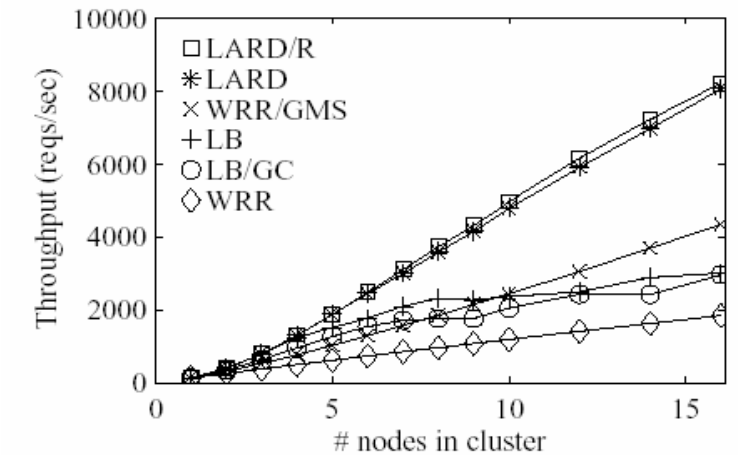
- CPU : Pentium II 300 Mhz
- Network : 100 Mbps LAN
- Disk : WD Enterprise
- Caching : GDS
- Filesystem : Berkeley FFS
- Input :Rice & IBM trace
- Strategies : WRR, LB-Locality, LARD /R

PURDUE UNIVERSITY

35

Simulation Results

THROUGHPUT RESULTS

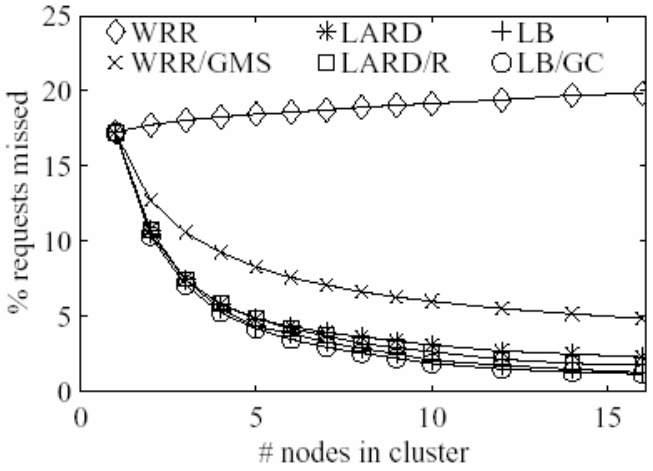


PURDUE UNIVERSITY

36

Simulation Results

MISS RATE

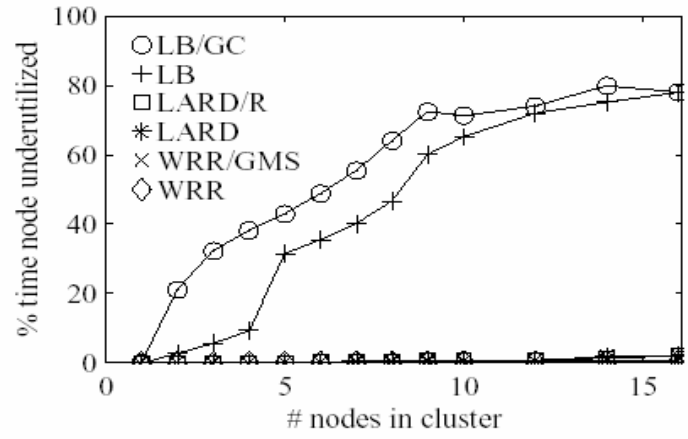


PURDUE UNIVERSITY

37

Simulation Results

LOAD BALANCING

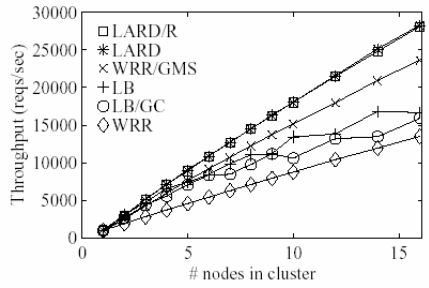


PURDUE UNIVERSITY

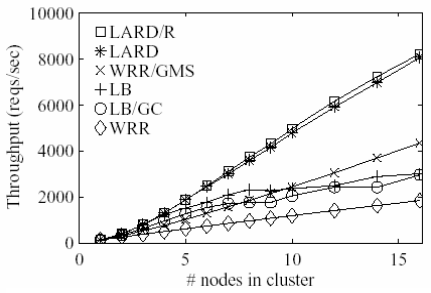
38

Simulation Results

IBM VERSUS RICE WORKLOAD



IBM

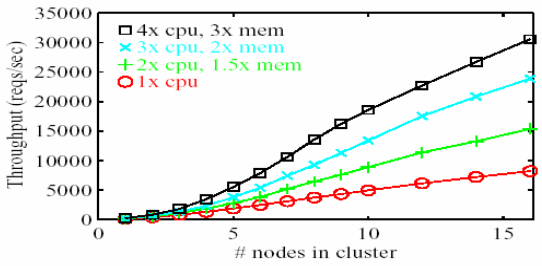


RICE

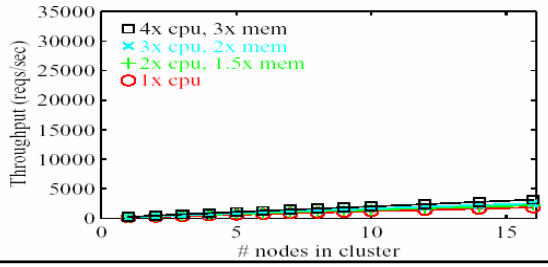
PURDUE UNIVERSITY

Simulation Results

SENSITIVITY TO CPU (LARD & WRR)



LARD/R

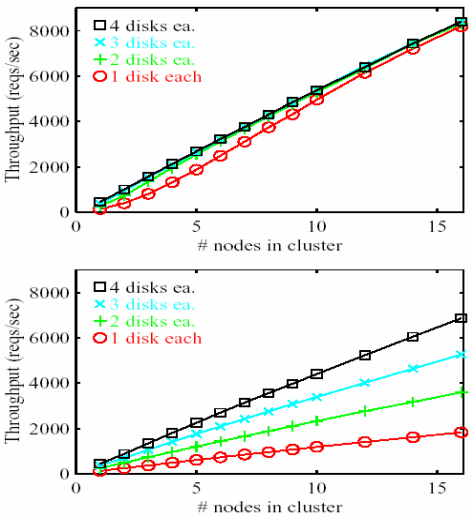


WRR

PURDUE UNIVERSITY

Simulation Results

SENSITIVITY TO EXTRA DISKS (LARD & WRR)



LARD/R

WRR

PURDUE UNIVERSITY

41

Experimental Results

ENVIRONMENT

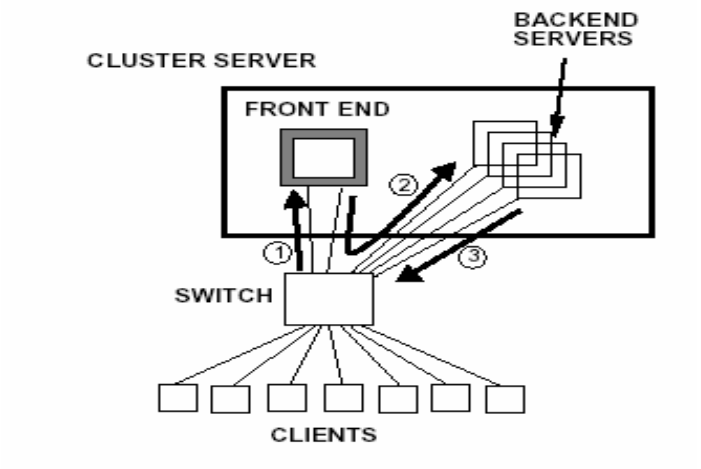
- Pentium II 300 Mhz 128 MB
- 100 Mbps Fast Ethernet
- FreeBSD 2.2.5
- Apache 1.2.4
- Rice Trace

PURDUE UNIVERSITY

42

Experimental Results

TESTBED

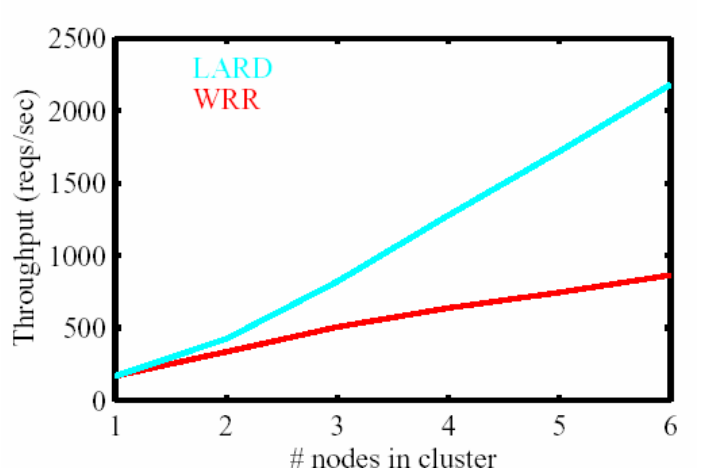


PURDUE UNIVERSITY

43

Experimental Results

THROUGHPUT



PURDUE UNIVERSITY

44

Issues

HOW & WHAT TO IMPROVE?

- WHAT
  - If back-end serving target x fails ?
    - Replicate (in the paper)
  - If load on all nodes rises to $2*T_{high}$?
    - Admission Control (in the paper)
  - **If the front-end is overwhelmed?**
    - **Distributed Distributor (follow up paper)**
  - **If the workload is primarily dynamic content?**
    - **LARD for Dynamic Content (attempt by students)**
  - If acting as a proxy is too much overhead for the front-end?
    - TCP Handoff not Splicing (in the paper)
  - **If connections are persistent**
    - **P-HTTP support [ARON99] (follow up paper)**

PURDUE UNIVERSITY

Optimizations

DISTRIBUTED DISTRIBUTOR – TWO SCHEMES

Why we need to do this.

We need this

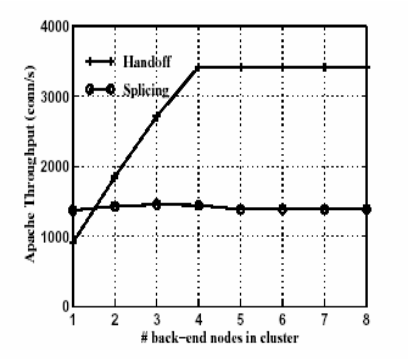


Figure 2: Throughput, 6 KB requests

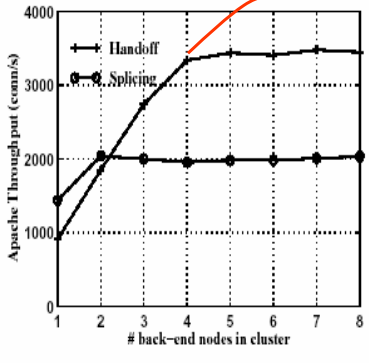
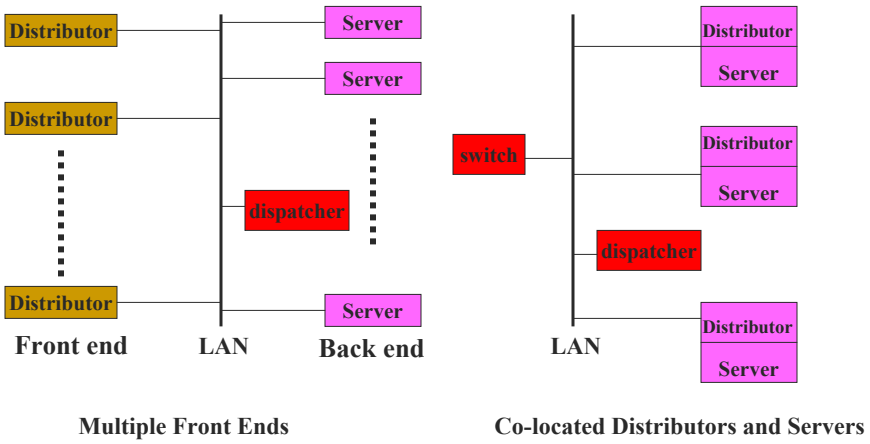


Figure 4: Throughput, IBM trace

PURDUE UNIVERSITY

Optimizations

DISTRIBUTED DISTRIBUTOR – TWO SCHEMES



Multiple Front Ends

Co-located Distributors and Servers

PURDUE UNIVERSITY

47

Optimizations

DISTRIBUTED DISTRIBUTOR

Multiple front ends mechanism:

1. Poor load balancing
2. Efficient partitioning of cluster nodes into either front end or back end nodes depends upon the workload (unpredictable), which may lead to low cluster utilization.

Co-located distributors and servers:

1. Cluster utilization is high & distributor bottleneck eliminated.
2. Layer 4 switch employed so scalability increased.
3. Centralized switch balances the load on each distributor

Overall disadvantages: A little latency incurred.

Overall Advantages: Scalability is very high

PURDUE UNIVERSITY

48

Optimizations

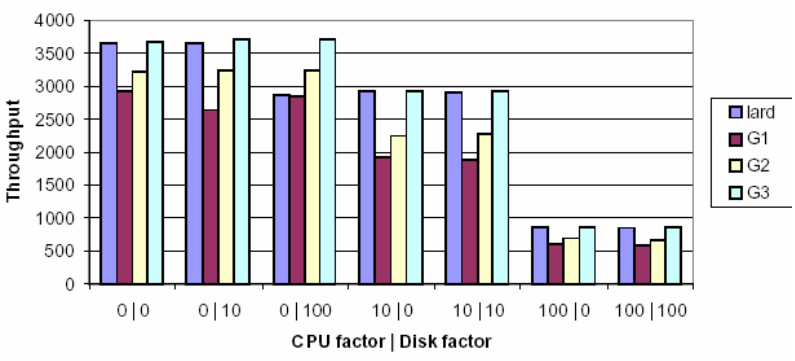
DYNAMIC CONTENT

- Serving dynamic content requires
  - More CPU time and disk bandwidth
  - More complex
- What can be done?
  - DLoad = $W_{cpu*}CPU_{utilization} + W_{disk*}Disk_{utilization}$
  - Reassign target documents when
    - node is overloaded (connections > $2T_{high}$)
    - node is overloaded and ∃ node w. connections < T_{low}
    - node DLoad is high and “under-dloaded” node exists
- Shown to perform better than LARD

PURDUE UNIVERSITY

Optimizations

DYNAMIC CONTENT



PURDUE UNIVERSITY

Optimizations

PERSISTENT CONNECTIONS

- Problem :
  - LARD operates at the granularity of TCP connections
  - With HTTP/1.1, multiple HTTP requests may arrive on a single TCP connection
  - Result (2 problems)
    - All requests arriving on a given connection must be served by a single back-end node
    - A back-end node may receive requests that it cannot serve if it is a specialized node

PURDUE UNIVERSITY

Optimizations

PERSISTENT CONNECTIONS - SOLUTIONS

- Redesign handoff to support HTTP/1.1 by allowing the front-end to migrate a connection between back-end nodes
- New handoff mechanism : back-end request forwarding
  - front-end hands off client connections to an appropriate back-end node using the TCP single handoff protocol
  - If request arrives on a p-connection that can/should not be served by the current back-end A, the connection is *not* handed off to another back-end node B
  - front-end informs *A* which backend (B) should serve the request.
  - *A* then requests the content directly from *B*, and forwards the response to the client on its client connection

PURDUE UNIVERSITY

Optimizations

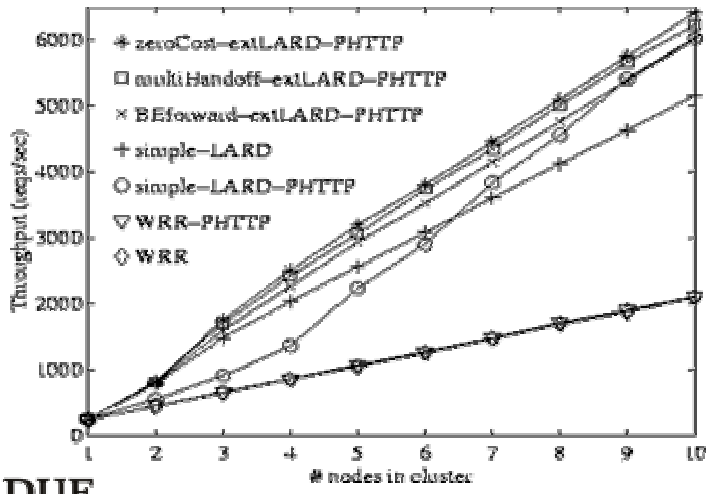
PERSISTENT CONNECTIONS – EXTENDING LARD

- Factors to consider
  - Choice of assignment constrained by previous choice
  - Reassignment overhead should be considered
- Extended LARD
  - First request on a p-connection assigned backend using original LARD policy
  - For subsequent requests
    - If the target is cached at the connection handling node or if the disk utilization on the connection handling node is low (less than 5 queued disk events), then the request is assigned to the same
    - Else, the three cost metrics (locality, load, replacement) are computed over the connection handling node and any other back-end nodes that have the target cached. The request is then assigned to the node that yields the minimum aggregate cost

PURDUE UNIVERSITY

Optimizations

PERSISTENT CONNECTIONS – RESULTS



PURDUE UNIVERSITY

Conclusions

PERFORMANCE EVALUATION OVERWIEW

- LARD paper compares SLB/WRR and LB with LARD approaches:
  - simulation study
    - small Rice and IBM web server logs
    - tweak simulation parameters to achieve desired result?
  - Nodes have small memories with *greedy-dual* replacement.
  - WRR combined with global cache-sharing among servers (GMS).
    - WRR/GMS is global cache LRU with duplicates and cache-sharing cost.
    - LB/GC is global cache LRU with duplicate suppression and no cache-sharing cost.

PURDUE UNIVERSITY

Conclusions

PERFORMANCE EVALUATION RESULTS

- 1. WRR has the lowest cache hit ratios and the lowest throughput.
  - There is much to be gained by improving cache effectiveness.
- 2. LB achieve slightly better cache hit ratios than LARD.
  - WRR/GMS lags behind because of duplicates.
- 3. The caching benefit of LB is minimal, and LB is almost as good as LB/GC.
  - Locality- request distribution induces good cache behavior at the back ends: global cache replacement adds little.
- 4. Better load balancing in the LARD strategies dominates the caching benefits of LB.
  - LARD/R and LARD achieve the best throughput and scalability; LARD/R yields slightly better throughput.

PURDUE UNIVERSITY

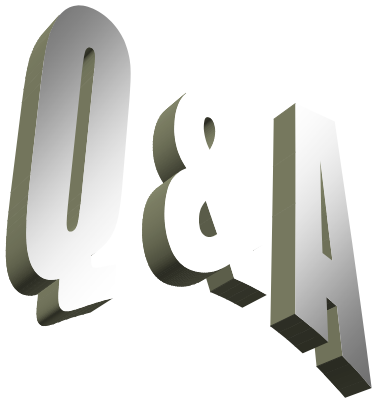
Conclusions

ISSUES AND QUESTIONS

- 1. LB (URL switching) has great cache behavior but lousy throughput.
  - Why? Underutilized time results show poor load balancing.
- 2. WRR/GMS has good cache behavior and great load balancing, but not-so-great throughput.
  - Why? How sensitive is it to CPU speed and network speed?
- 3. What is the impact of front-end caching?
- 4. What is the effectiveness of bucketed URL hashing policies?
  - E.g., Foundry: hash URL to a bucket, pick server for bucket based on load.
- 5. Why don't L7 switch products support LARD? Should they?
  - [USENIX 2000]: use L4 front end; back ends do LARD handoff.

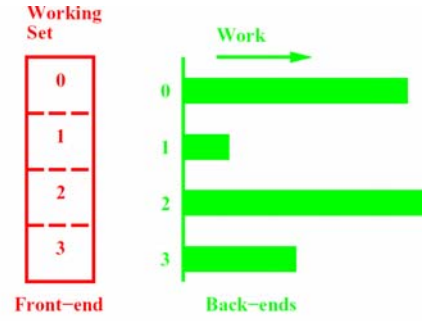
PURDUE UNIVERSITY

Questions & Ambiguous Responses

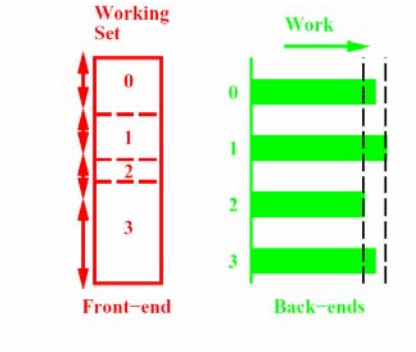


PURDUE UNIVERSITY

LARD motivation



Goal: "equally" divide working set

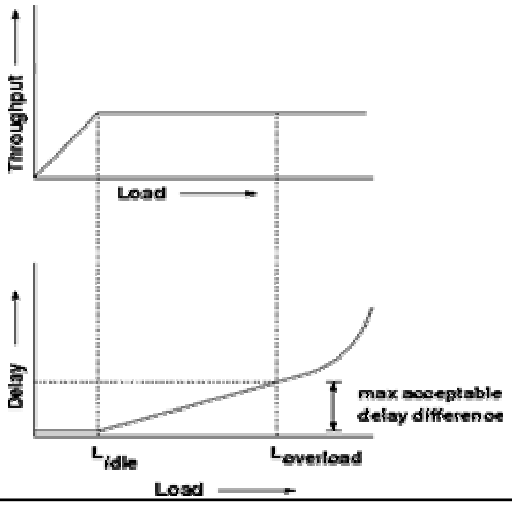


Goal: divide working set to balance work

PURDUE UNIVERSITY

59

LARD Intuition

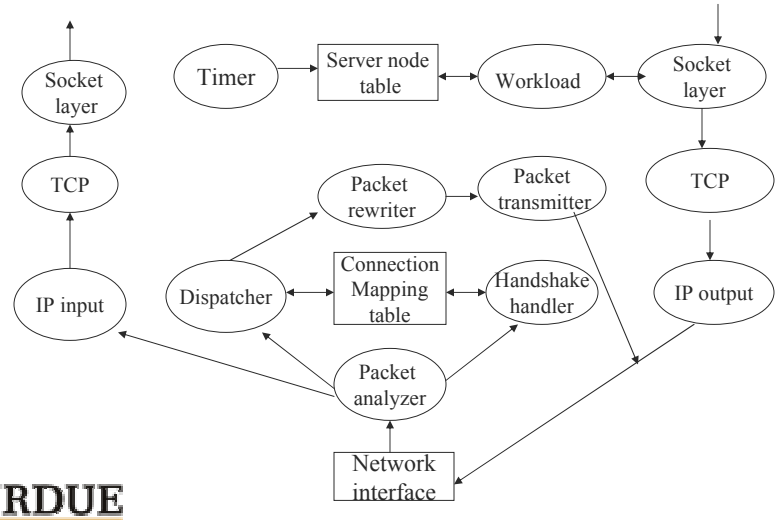


PURDUE UNIVERSITY

60

Detailed Idea

IMPLEMENTATION SCHEMATIC



PURDUE UNIVERSITY

61