The Effectiveness of Request Redirection on CDN Robustness

Limin Wang, Vivek Pai and Larry Peterson

Presented by:

Eric Leshay

Ian McBride

Kai Rasmussen

Outline

- Introduction
- Redirection Strategies
- Methodology
- Normal Load Results
- Flash Crowd Results
- Conclusion

CDNs

- To achieve better performance, networks can be built using redundant resources
  - Content Distribution Networks (CDN)
- Improves -
  - Response Time
    - Cumulative latencies
  - System Throughput
    - Average number of requests satisfied every second

CDN Distribution Factors

- Network Proximity
  - Minimizes response time
- Balance System Loads
  - Improves system throughput
- Locality
  - Select server with page already in cache
  - Overall improvements

CDN Model

- Server Surrogate
  - Caches page normally kept on a set of backend servers
  - Uses replication to improve response time and system throughput
- Uses request redirectors
  - Transparent system to get user to a file without user knowing about any replication

Redirector Mechanisms

- DNS server augmentation
  - Site Level
  - Caching problems avoided with short expiration times
- Server Redirection
  - HTTP Redirect Response
    - Adds extra round trip time
    - Consumes bandwidth

Redirector Mechanisms

- Router or proxy rerouting
  - Rewrite outbound request
  - HTTP Redirect
  - Proxies on edge of server
    - Approximate load Information
    - Identifiable client population

Hashing Schemes

- Maps URLs to range of values
- Modulo Hashing
  - URL is hashed, n % (# of servers)
  - Must change modulus as number of servers change
- Consistent Hashing
  - URL and names of servers hashed in long circular list
  - URL assigned to closest server in list

Hashing Schemes

- Highest Random Weight
  - Hashes URL and server names by random weights, and sorts result
  - List is traversed to find appropriate server
  - More computation than consistent hashing

Request Redirection Strategies

- Random
  - Request randomly sent to a server surrogate
  - ‘baseline’ to determine reasonable performance
- Static Server Set
  - Assigns a fixed number of server replicas to each URL
  - Improves Locality
- Load-Aware Static Server Set
  - Redirects based on approximated load information
- Dynamic Server Set
  - Adjusts number of replicas for better locality and load balancing
- Network Proximity
  - Favors shorter network paths

Static Server Set

- Replicated Consistent Hashing (R-CHash)
  - Number of replicas is fixed but configurable
  - URL and replicas hashed to circular space
  - Redirector assigns a request to a replica for the URL
- Replicated Highest Random Weight (R-HRW)
  - Uses HRW to hash URL and replicas
  - Replicas for each URL decided by top N servers from the ordered weighted list

Load-Aware Static Server Set

- Redirectors maintain estimates of server load
- Finds least loaded server for redirection
- Load-Aware counter parts of R-CHash and R-HRW
  - LR-CHash
  - LR-HRW

Dynamic Server Set

- Dynamically adjusts the number of replicas
- Introduces two new algorithms
  - Coarse Dynamic Replication (CDR)
  - Fine Dynamic Replication (FDR)
- Factors both load and locality into decision combined with a dynamic set of replicas

Coarse Dynamic Replication

- Uses HRW hashing to create an ordered list of servers
- Coarse-grained load information is used to select first non-busy server
  - Number of active communications used to approximate load level
  - Can be combined with response latency, bandwidth consumption and other factors

CDR Code

```
find_server(url, S) {
    foreach server s_i in server set S,
        Weight_i = hash(url,address(s_i));
    Sort weight;
    foreach server s_j in decreasing order of weight_j {
        If satisfy_load_criteria(s_j) then {
            targetServer = s_j;
            Stop search;
        }
    }
    If targetServer is not valid then
        targetServer = server with highest weight;
    Route request url to targetServer;
}
```

Fine Dynamic Replication

- Uses URL popularity to decrease unnecessary replication
- Introduces a walk length to indicate the number of servers that should be searched
- If all servers are busy the walk length is increased
- Keep track of modified time. Walk length is decreased after a long time unmodified

FDR Code 1

Find_server(*url*, *S*){

walk_entry = walkLenHash(*url*);

w_len = walk_entry.length;

foreach server s_i in server set *S*,

weight$_i$ = hash(*url*,address(s_i));

sort *weight*;

$s_{candidate}$ = least-loaded server of top *w_len* servers;

if *satisfy_load_criteria*($s_{candidate}$) then {

targetServer = $s_{candidate}$;

if (*w_len* > 1

&& (timenow() - walk_entry.lastUpd > chgThresh)

walk.entry.length--;

}

FDR Code 2

```
else {
        foreach rest server s_j  in decreasing weight order {
                if satisfy_load_criteria(s_j ) then {
                        targetServer = s_j;
                        stop search;
                }
        }
        walk_entry.length = actual search steps;
}
if walk_entry.length changed then
        walk_entry.lastUpd = timenow();
if targetServer is not valid then
        targetServer = server with highest weight;
route request url to targetServer;
}
```

Network Proximity

- Map addresses to a geographic region. Select servers within a specific region
  - Find closer servers
- Use *ping* and *traceroute* to measure topological location
- Three Network Proximity strategy
  - NP-FDR
  - NPR-CHash
  - NPLR-CHash

Methodology: Simulation

- Network & OS/server combo
  - NS-2: packet-level simulator
    - Tests TCP implementations
  - Logsim: server cluster simulator
    - Simulates CPU processing, memory usage, and disk access

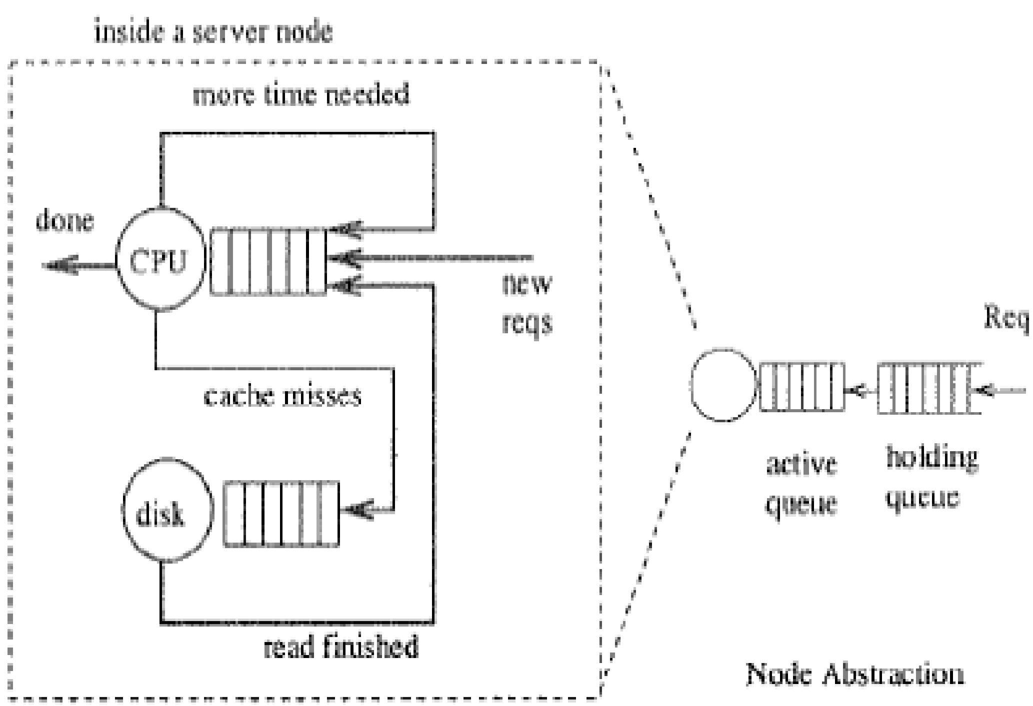
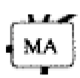


Figure 3: Logsim Simulator

Methodology: Network Topology

- NSFNET backbone network T3 topology
  -  server surrogates
  -  client hosts
  -  regional routers
  -  backbone routers (with location)

  - 64 servers
  - 1,000 client hosts
  - 1,100 nodes

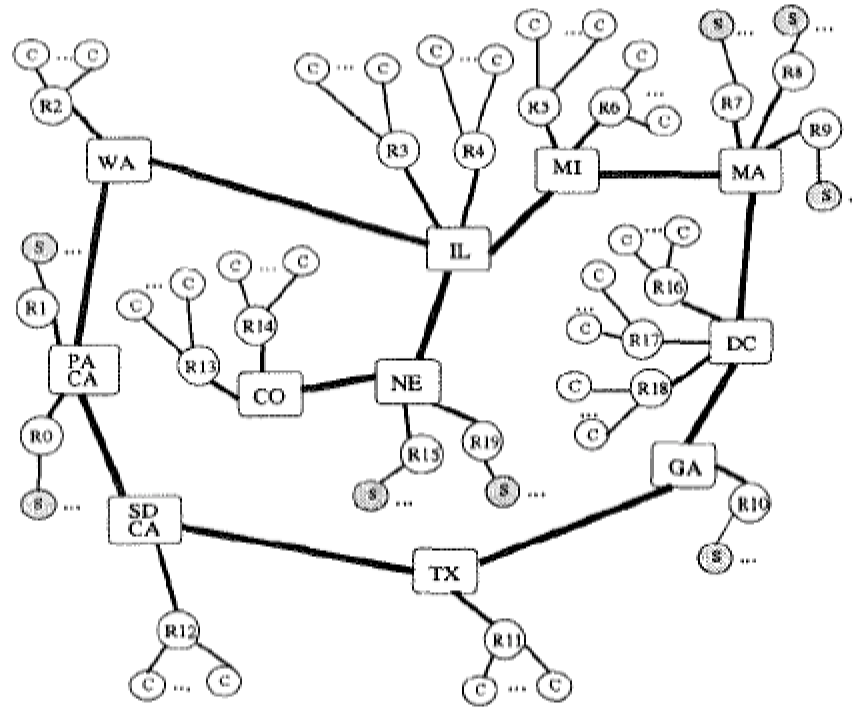
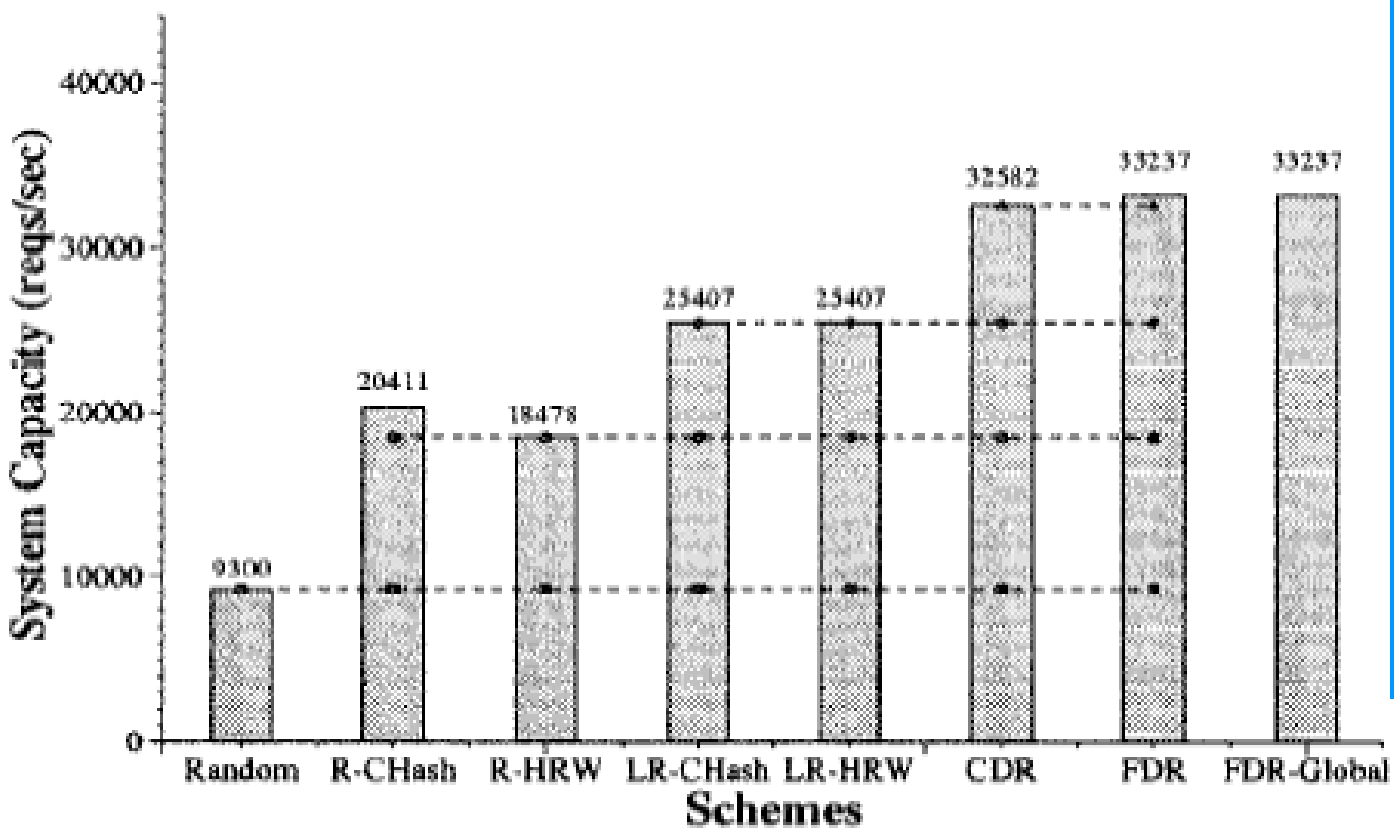


Figure 4: Network Topology

Results: Normal Load

- Charts shown are for 64 server case.
- Optimal Static Replication
  - Performance R-CHash and R-HRW influenced by number of replicas.
  - For 2 to 64 replicas:
    - Increasing replicas help load balance; improves throughput
    - Too many replicas will hinder throughput, as replica working set causes more server disk activity.
    - 10 replicas determined to be optimal number.

Results: Normal Load: Capacity



- R-CHash 119% better than random.
- R-HRW 99% better than random.
- LR-CHash and LR-HRW 173% better than random.
- CDR and FDR 250% better than random.

Results: Normal Load: Server Resource Utilization

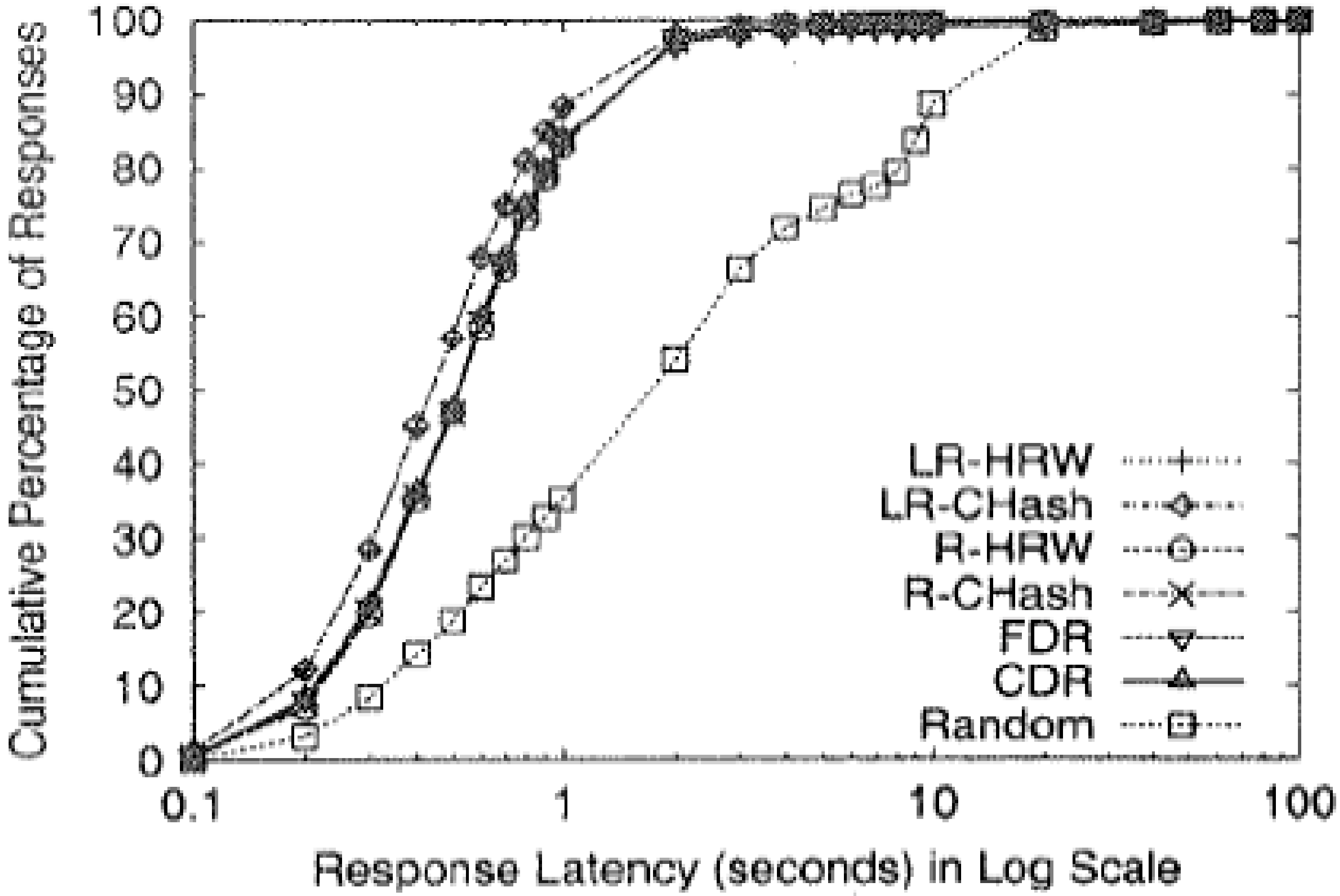
"With faster simulated machines, we expect the gap between the dynamic schemes and the others to grow even larger."

Utilization Scheme	CPU (%)		DISK (%)	
	Mean	Stddev	Mean	Stddev
Random	21.03	1.36	100.00	0.00
R-CHash	57.88	18.36	99.15	3.89
R-HRW	47.88	15.33	99.74	1.26
LR-CHash	59.48	18.85	97.83	12.51
LR-HRW	58.43	16.56	99.00	5.94
CDR	90.07	11.78	36.10	25.18
FDR	93.86	7.58	33.96	20.38
FDR-Global	91.93	11.81	17.60	15.43

Table 2: Server Resource Utilization at Overload

- Hash schemes utilize disk more, processor less.
- Dynamic schemes (CDR and FDR) utilize processor more, disk less.

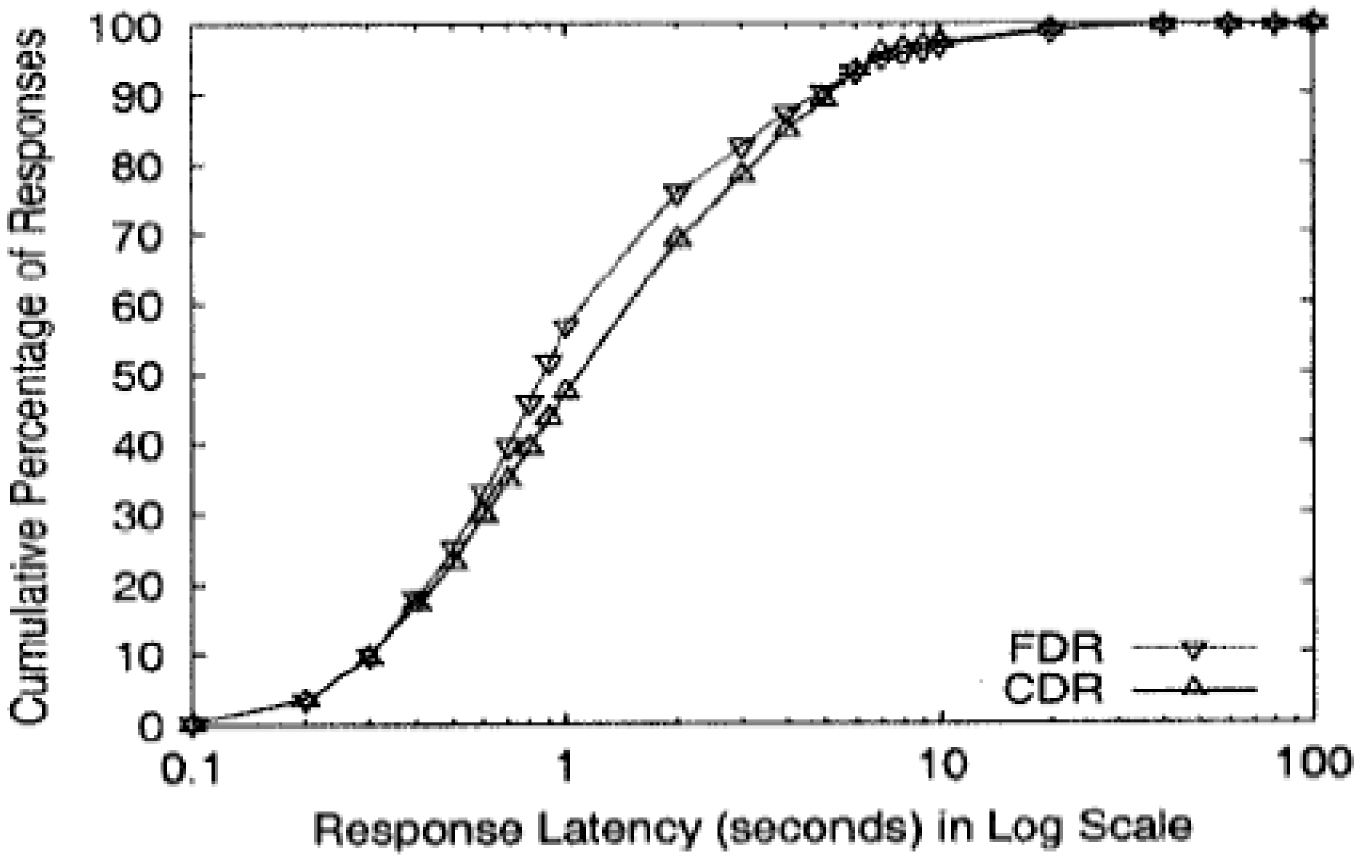
Results: Normal Load: Latency



(a) Random's limit: 9,300 req/s

- Dynamic schemes (CDR, FDR) similarly outperform hash schemes at low response loads.
- Dynamic and static schemes serve large files roughly the same, since large files are replicated less under CDR/FDR.

Results: Normal Load: Latency



(d) CDR's limit: 32,582 req/s

- FDR outperforms CDR at very high response rates for files of median sizes.

- Small files are served at roughly the same rate by both schemes. Large files still suffer from under – replication.

Results: Normal Load: Scalability

- Experiments repeated for 8 to 128 servers.

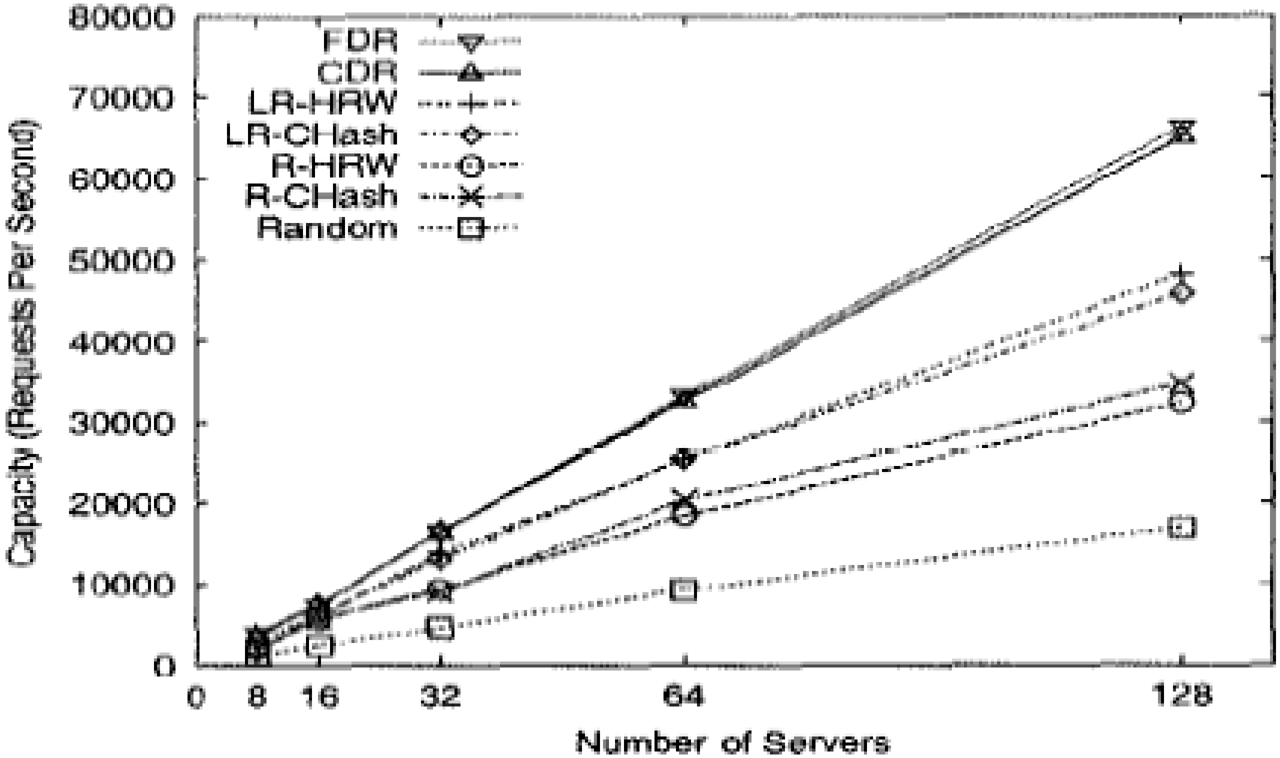


Figure 7: System Scalability under Normal Load

- Linear growth meaning systems scale well.
- Server router to backbone router link bandwidth doubled for 128 server case.

Behavior under Flash Crowds

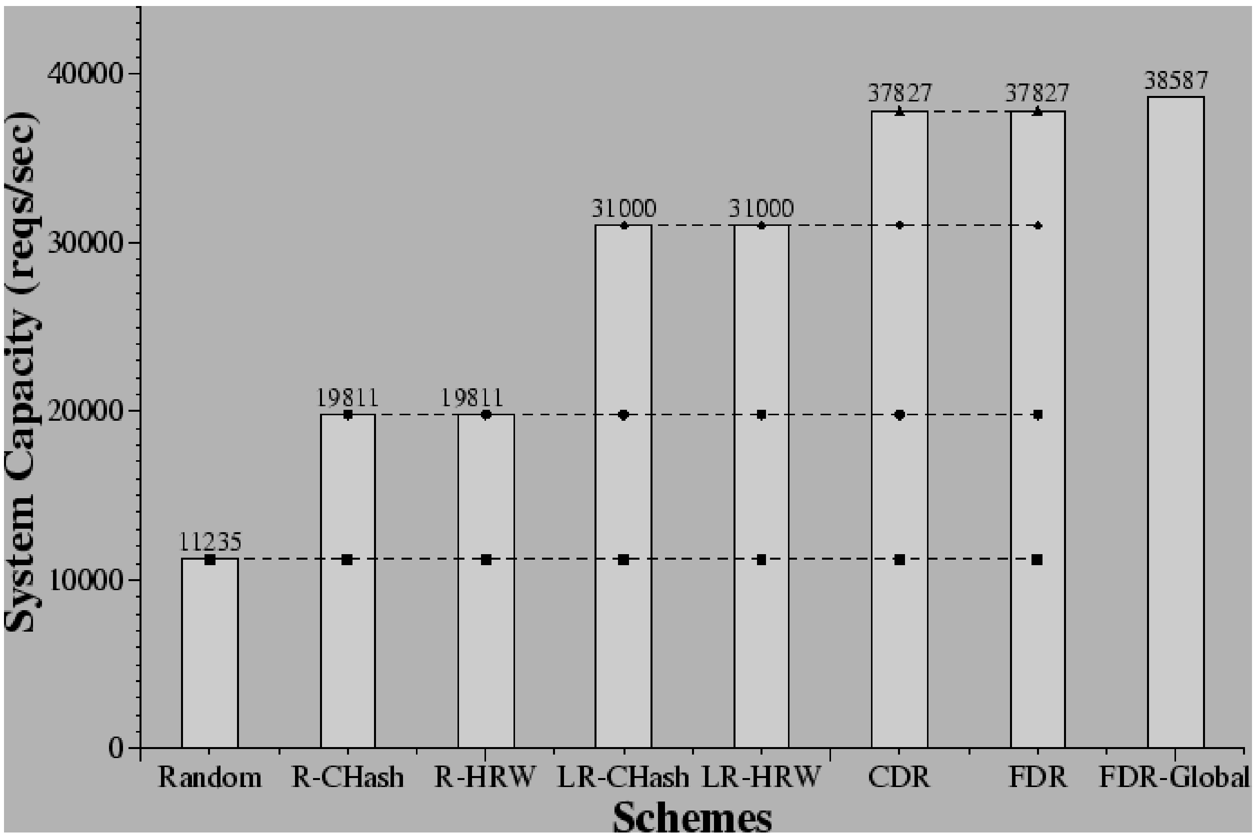
- Simulate the performance of CDNs under a flash crowd or DDoS attack
- Measured performance by:
  - Capacity – requests/second
  - Latency – response time in seconds
  - Scalability – requests/second

Flash Crowd Setup

- System Capacity – requests/second, latency
  - 1000 clients – 25% intensive requesters
  - Intensive requesters download a URL of 6kb from a predetermined list over and over
  - Clusters of 64 servers

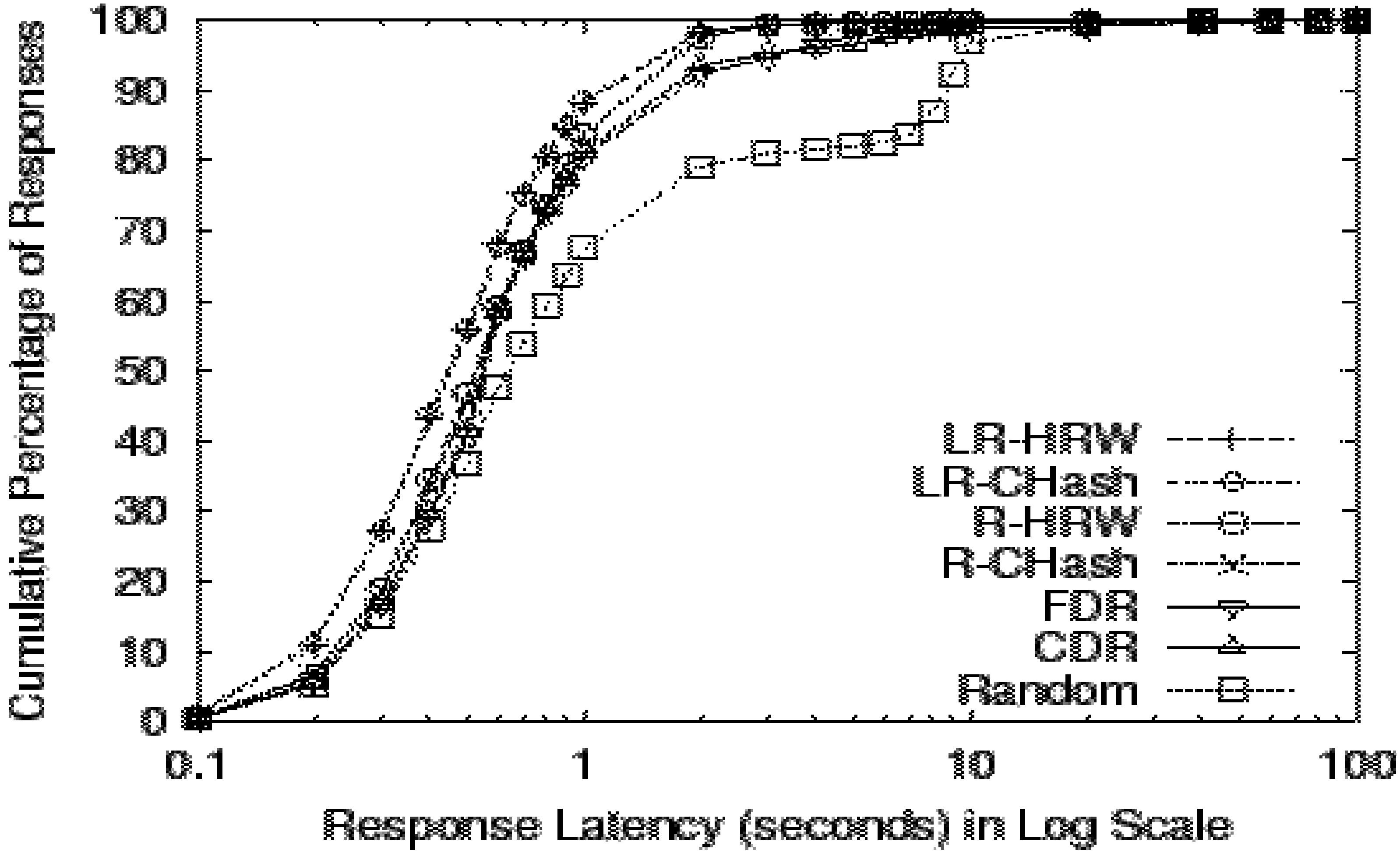
- Scalability - requests/seconds
  - Varying cluster size from 32 – 128 servers

System Capacity – Flash Crowd



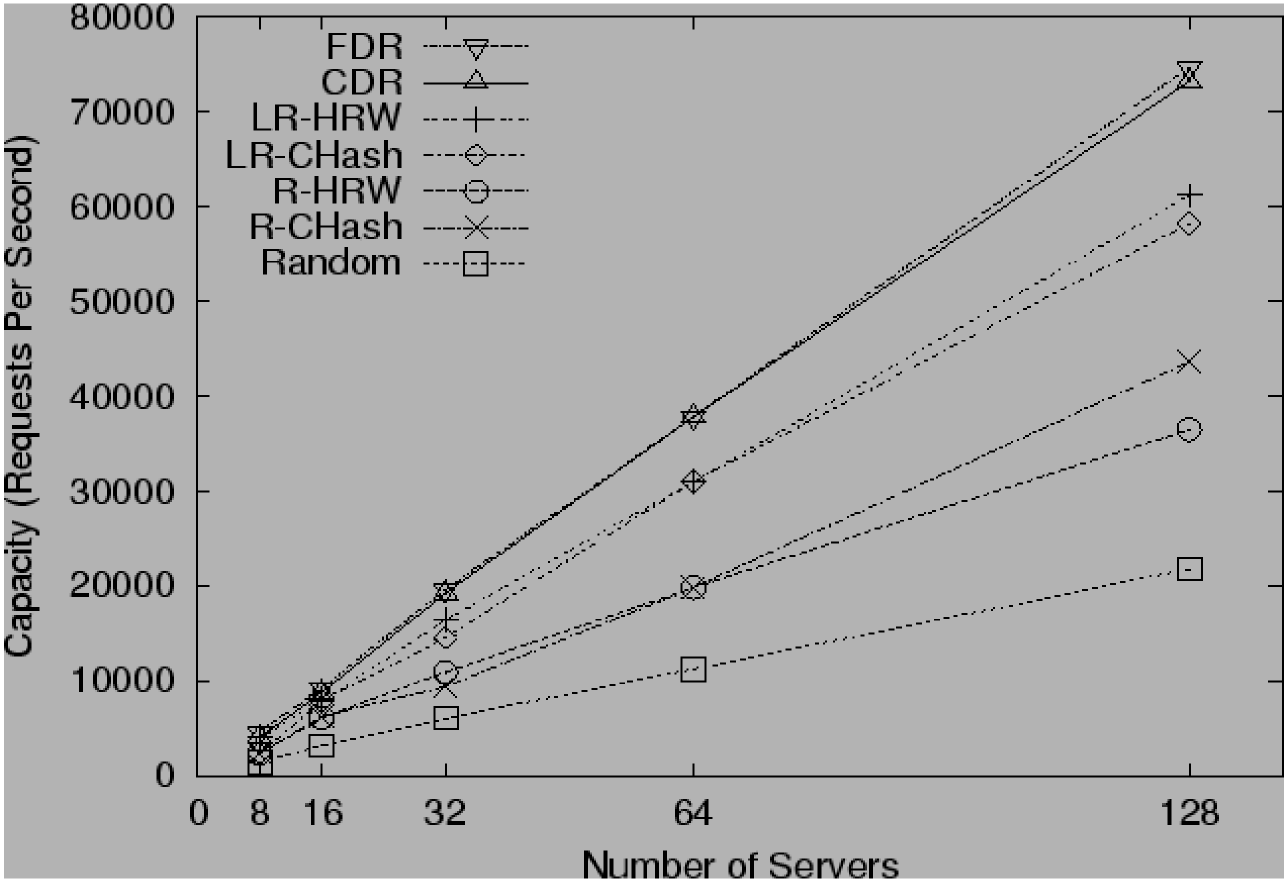
- FDR's benefit has grown to 91% from 60% over R-HRW and LR-CHash during the flash crowd scenario

- Random has the worst latency, LR-CHash and LR-HRW have the best latency.



- In a direct comparison of FDR and CDR, FDR proves to have the best latency

Scalability Results – Flash Crowd

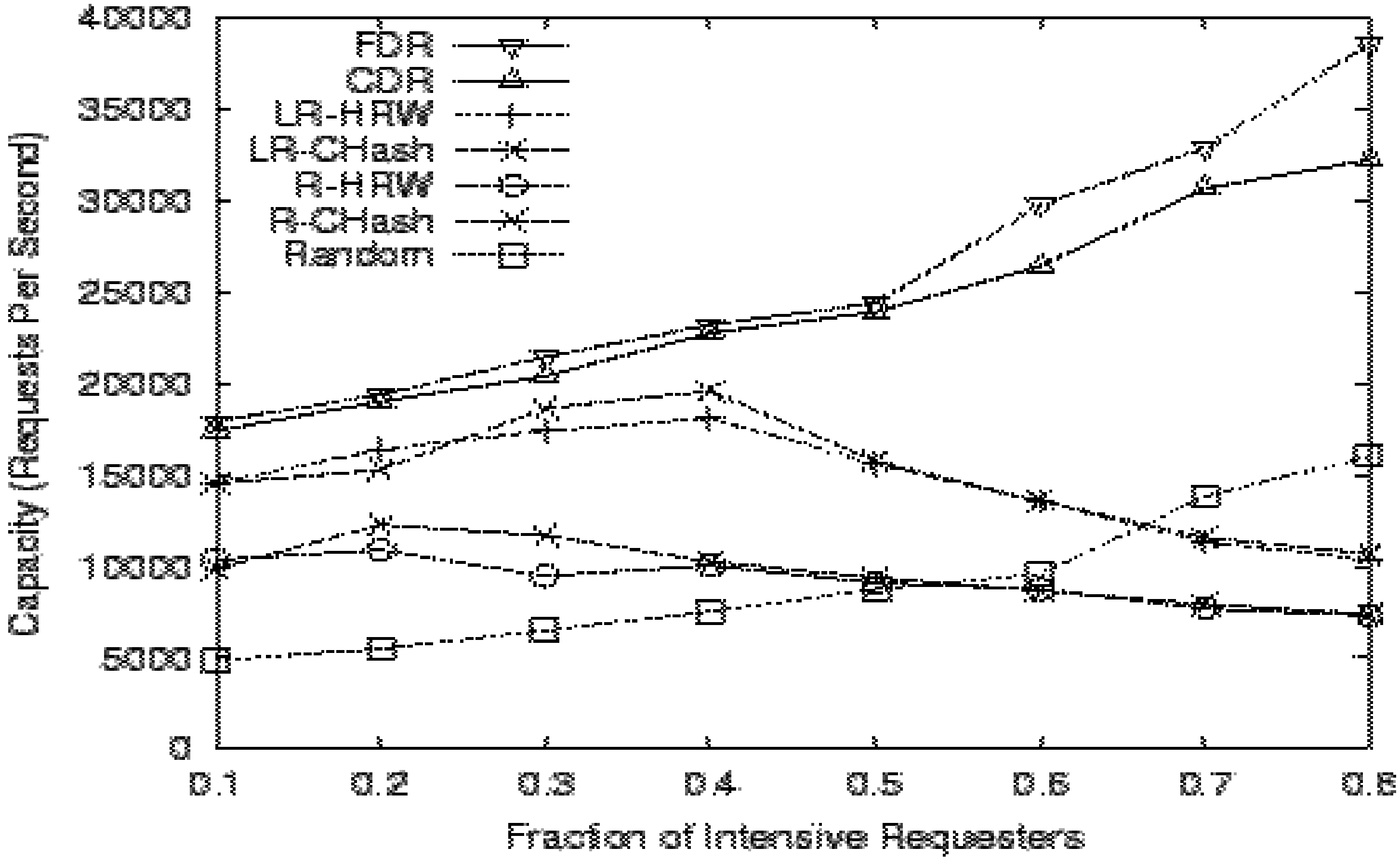


- All the algorithms scale linearly
- Similar results to the trial under normal load

More Flash Crowd Tests

- Flash Crowds setup
  - 1 hot URL of 1kb
  - 10 hot URLs of 6kb
- Test parameters
  - Vary intensive requesters from 10% - 80%
  - Vary cluster size from 32 to 64 servers
  - Measured requests/second

- FDR and CDR are able to adapt to flash crowds
- All other algorithms perform worse than random during the flash crowd
- Tested with 1 hot URL and 10 hot URLs



Proximity Comparison

Table 7: Proximity's Impact on Response Latency under Flash Crowds. μ -- Mean, σ -- Standard Deviation.

Req Rate	*11,235 req/s*				*14,409 req/s*				*30,090 req/s*				*34,933 req/s*			
Latency	μ	50%	90%	σ	μ	50%	90%	σ	μ	50%	90%	σ	μ	50%	90%	σ
Random	2.37	0.64	8.57	5.29												
NPR-CHash	0.61	0.42	1.15	1.76	0.63	0.41	1.08	2.34								
R-CHash	0.73	0.53	1.45	2.10	0.73	0.52	1.38	2.50								
NPLR-CHash	0.53	0.36	0.90	1.75	0.55	0.35	0.91	2.29	1.29	0.61	2.65	3.94				
LR-CHash	0.62	0.45	1.15	1.70	0.64	0.44	1.13	2.56	0.90	0.49	1.73	3.44				
NP-FDR	0.70	0.50	1.45	1.68	0.66	0.45	1.34	1.63	0.81	0.47	1.64	2.55	0.99	0.51	1.92	3.26
FDR	1.22	0.55	1.81	5.71	1.07	0.54	1.67	5.47	1.60	0.66	3.49	5.90	1.84	0.78	4.15	6.31

- Proximity can benefit latency, but may hurt capacity. This is the case with NPLR-CHash

Heterogeneity

- Random and R-CHash cannot determine speed of links
- LR-CHash and FDR are able to assign requests fairly between slow and fast links

Table 8: Capacity (reqs/sec) with Heterogeneous Server Bandwidth,

	Portion of Slower Links					
Redirection	Normal Load			Flash Crowds		
Schemes	0%	10%	30%	0%	10%	30%
Random	9300	8010	8010	11235	8449	8449
R-CHash	20411	7471	7471	19811	7110	7110
LR-CHash	25407	23697	19421	31000	26703	22547
FDR	33237	31000	25407	37827	34933	29496

Large File Effects

Table 9: Response Latency with Special Large File Handling, Normal Load. μ -- Mean, σ -- Standard Deviation.

Req Rate	*9,300 req/s*				*18,478 req/s*				*25,407 req/s*				*32,582 req/s*			
Latency	μ	50%	90%	σ	μ	50%	90%	σ	μ	50%	90%	σ	μ	50%	90%	σ
LR-CHash	0.68	0.44	1.17	2.50	0.87	0.51	1.82	2.74	1.19	0.60	2.47	3.79				
LR-HRW	0.68	0.44	1.18	2.50	0.90	0.51	1.89	3.13	1.27	0.64	2.84	3.76				
CDR	1.16	0.52	1.47	5.96	1.35	0.55	1.75	6.63	1.86	0.63	4.49	6.62	2.37	1.12	5.19	7.21
CDR-T-R	0.78	0.52	1.43	2.77	0.76	0.52	1.40	2.80	1.05	0.57	1.90	3.06	1.58	0.94	3.01	3.55
CDR-T-S	0.74	0.52	1.43	2.17	0.72	0.52	1.38	2.44	1.01	0.56	1.93	2.96	1.53	0.68	3.69	4.18
FDR	1.10	0.52	1.48	5.49	1.35	0.54	1.64	6.70	1.87	0.62	3.49	6.78	2.22	0.87	4.88	7.12
FDR-T-R	0.78	0.52	1.43	2.77	0.75	0.52	1.40	2.82	1.01	0.57	1.87	2.98	1.39	0.77	2.82	3.68
FDR-T-S	0.74	0.52	1.43	2.17	0.72	0.52	1.37	2.55	0.98	0.56	1.84	2.95	1.41	0.63	2.88	3.88

- Threshold set where files > 530kb sent to a separate server

Conclusion

- Improved Redirect Algorithms lead to more robust CDN systems
- FDR allows a 60-91% greater load than previously published systems
- FDR provides a mechanism for defending against flash crowds or Distributed Denial of Service Attacks

Questions/Discussion