FAB-Cache: A Programmable Multi-Site Caching Service for Remote Dataset Retrieval Using FABRIC

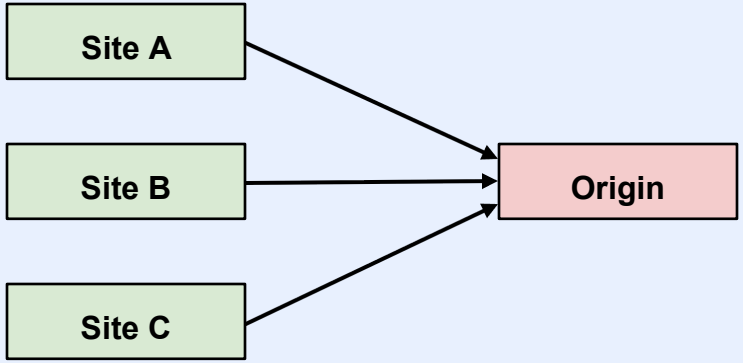
SC26 INDIS Workshop

Pinyi Shi, Vaiden Logan, Dakota Morgan, James Griffioen, Zongming Fei, Kenneth Calvert

University of Kentucky

The Problem

- Scientific datasets are huge (GBs to 100s of TBs)
- Many sites download the same data
- Every copy crosses the Internet
- The origin server can be overloaded
- Wide-area bandwidth is wasted



The same data downloaded repeatedly from origin

Existing Approaches Fall Short

Approach	Problem
Commercial CDNs	File size limits (512 MB to 5 GB at Cloudflare), little control over the path, pricing built for web traffic
Domain-specific systems (XRootD, OSDF)	Built for one community, need large dedicated infrastructure
Named Data Networking	New network architecture, applications must be rewritten

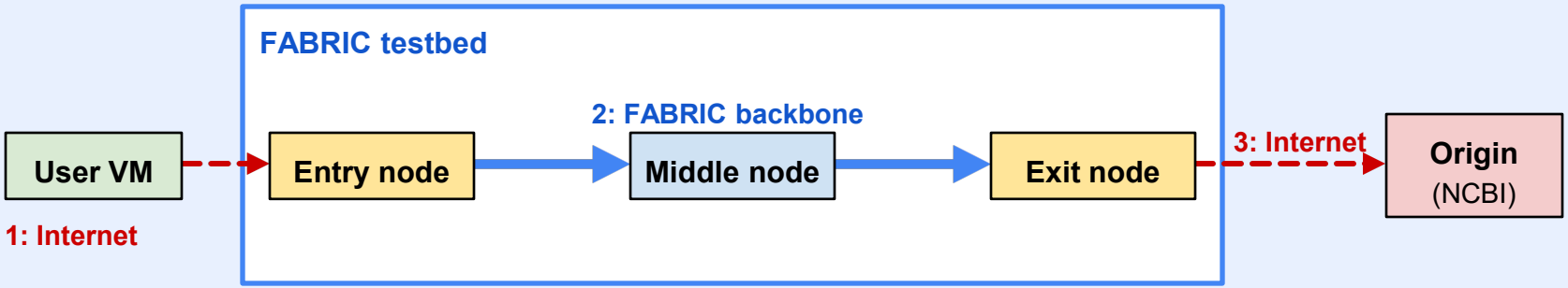
Why FABRIC?

- Programmable testbed with racks at 30+ sites
- A large amount of persistent storage per site
- Dedicated Terabit backbone links
- perfSONAR measurements between sites
- Standard protocols (HTTP, FTP)



Compute, storage, links and measurements in one place

Idea: Keep Data Off the Internet



- 1: The user picks the nearest caching node
- 2: The path through FABRIC comes from perfSONAR measurements
- 3: The exit node is close to the data source

How a Request Flows

User sends a curl command to the Flask app on the entry node. Flask adds an X-Proxy-Path header (the ordered list of node IPs), and NGINX forwards the request node by node.

Cache miss

- Each node checks its disk cache, then forwards the request
- The exit node fetches the file from the origin
- Every node stores the response on the way back

Cache hit

- The first node holding the file serves it from disk
- The request goes no further

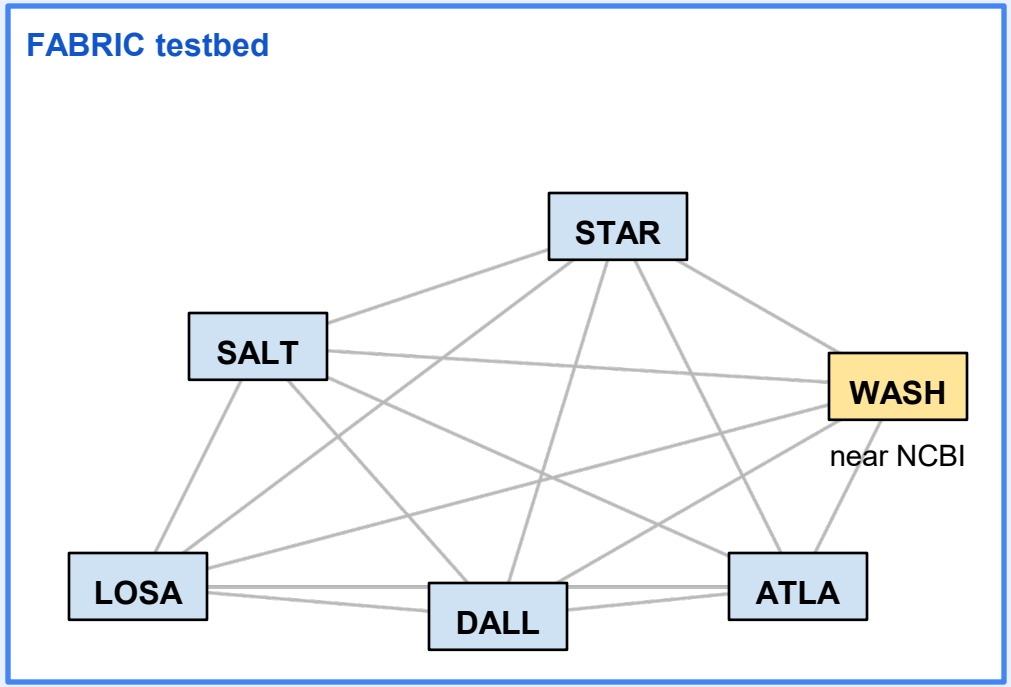
Choosing the Path

- perfSONAR tests throughput between all pairs of sites
- Pick the path whose slowest link is the fastest
- Example: LOSA to WASH goes LOSA, SALT, STAR, WASH
- No bandwidth reservation yet

Access and Monitoring

- Web portal: registered FABRIC users only, time-limited access
- Inside FABRIC: connect over fabnetV6
- Outside FABRIC: Rathole tunnel to the nearest node
- Prometheus and Grafana: cache usage and bandwidth
- An API lists which URLs are cached on each node

Prototype: Six FABRIC Sites



Setup

- 16 cores, 16 GB RAM, 500 GB disk
- 20 TB cache volume per node
- NCBI files: 1, 5, 50, 120 GB
- Direct vs. cache miss vs. hit
- Direct = public Internet route
- Three runs, mean throughput

Results: Download Throughput

Cache Hits beat direct Internet download in all 24 cases.

Direct: download straight from NCBI over the public Internet

Miss: first request through FAB-Cache: fetched from NCBI and cached

Hit: data already cached: served from FAB-Cache

	1 GB			5 GB			50 GB			120 GB		
	Direct	Miss	Hit	Direct	Miss	Hit	Direct	Miss	Hit	Direct	Miss	Hit
LOSA	48.8	87.1	**851**	51.1	90.6	**892**	48.8	64.8	**304**	50.5	58.4	**318**
SALT	60.4	90.9	**578**	62.4	93.5	**710**	61.9	61.7	**362**	63.2	72.8	**365**
STAR	155	156	**751**	174	180	**704**	83.4	83.9	**320**	138	129	**319**
DALL	86.5	148	**672**	92.8	136	**740**	73.4	63.5	**352**	91.8	83.6	**359**
ATLA	180	174	**952**	189	184	**821**	123	91.7	**369**	101	104	**368**
WASH	214	463	**680**	551	481	**774**	130	107	**338**	123	121	**341**

Download throughput in **MB/s** (mean of 3 runs).

What the Numbers Show

- Cache hits beat direct download in all 24 cases: 1.4× to 17× faster
- 1 and 5 GB hits: 578 to 952 MB/s (small enough for the Linux page cache)
- 50 and 120 GB hits: 304 to 369 MB/s (too big for RAM, read from disk)
- Far from NCBI, even a miss is faster: LOSA, 5 GB, 90.6 vs 51.1 MB/s
- Near NCBI, a miss is about the same as direct, sometimes lower
- The farther from the data, the more FAB-Cache helps

FABRIC Made This Practical

Open-source software

- NGINX (cache and proxy)
- Flask (requests)
- Rathole (tunnels)
- Prometheus + Grafana

From FABRIC

- VMs and 20 TB volumes at six sites
- Backbone links between sites
- perfSONAR measurements
- Slices and Jupyter notebooks

A multi-site service from standard software and testbed resources

Limitations and Future Work

Limitations

- Manual origin node mapping (single point of failure)
- Plain HTTP between nodes, public data only
- 7-day expiry, no invalidation
- No bandwidth reservation

Future work

- Backup origin nodes
- Automatic eviction
- Prefetching
- FABRIC QoS reservations
- All FABRIC sites

Updates Since Submission

- Facility ports: public IPs, so users connect directly
- Exit node picked automatically: lowest latency to the URL
- Token login: only members of one FABRIC project
- Each request has a token with a time limit, checked at the entry node
- Expanding to all FABRIC sites

Takeaways

- A programmable testbed can host a stateful, multi-site service
- Cache hits are 1.4× to about 17× faster than direct download
- Cache misses can match or beat direct download, most clearly far from the origin

Tutorial artifact: https://artifacts.fabric-testbed.net/artifacts/c8f4fa86-21cb-44fc-ab01-fb1a260e990f

Supported in part by NSF grants OAC-2029235 and CNS-2330891



QUESTIONS?