Topic 10: Secure Content Delivery

19/05/2014

Enrico Rotundo

Outline:

1. How can we deliver content to users?
    Is data replication secure?
2. What Content Delivery Network (CDN) is?
    Single server / CDN scheme
    Potential attacks
3. Secure Data Replication over Untrusted Hosts
    Dynamic queries on untrusted servers without state machine replication overhead?
4. Byzantine failures
    Byzantine-fault tolerant NFS
5. "Repeat and Compare"
    Features

How can we deliver content to users?

- high availability
- high performance

Data replication: fault tolerance and improved performance.

Is data replication secure?:

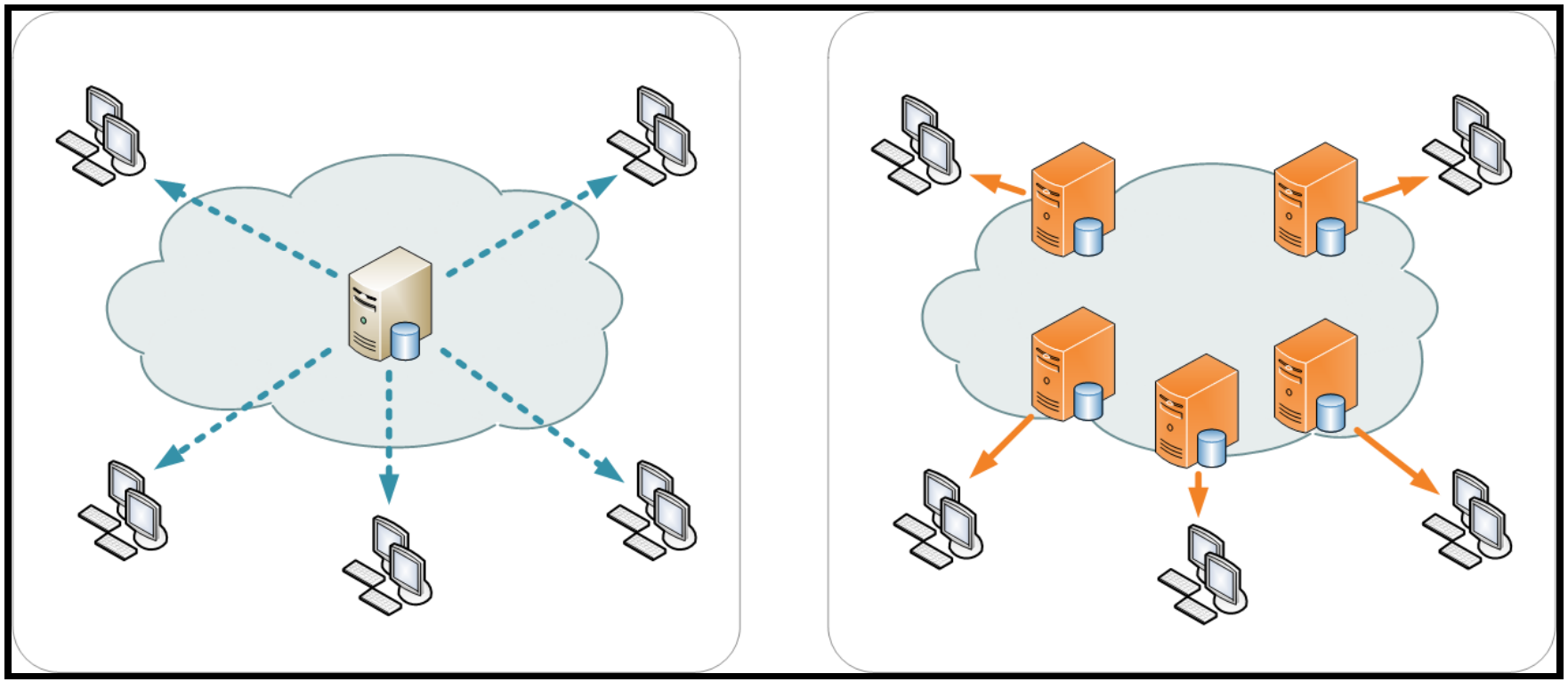
In Content Delivery Networks (CDN) datas are placed on hosts that are not directly controlled by the owner.

What Content Delivery Network (CDN) is?

A large distributed system of servers used to serve content to end-users with high availability and high performance.

- file sharing
- cooperative storage
- web caching
- web names look-up

Single server / CDN scheme



Existing CDNs:

- Na Kika
- CoralCDN (300-400 servers)
- CoDeeN

Cited CDNs properties:

- resources donated by research community
- trusted environment
- static content

Potential attacks:

- serving stale content (save bw)
- bypassing content transformations (reduce CPU usage)
- out-right modification (insert ads)

We need security mechanisms!

Secure Data Replication over Untrusted Hosts:

	Key idea	Pros & Cons
State signing	Data content is divided into subsets which are signed with a private key.	Only static data, restricted types of queries, state update on trusted servers.
State machine replication	Execute operation on untrusted hosts and accept it only when a majority agree.	Random queries, updates on untrusted servers, operations performed multiple times on different hosts. Latency dictated by the slowest server.

Dynamic queries on untrusted servers without state machine replication overhead?

Combining:

- only **statistical guarantees** on correctness
- background **auditing** mechanism

Assumption: Byzantine failures from untrusted components are rare.

Byzantine failures:

When components of a system fail in arbitrary ways.

Caused by:

- malicious attacks
- software errors

We need a Byzantine-fault tolerant system!

Byzantine-fault tolerant NFS:

Assumed n replicas,
works when no more then $\lfloor \frac{n-1}{3} \rfloor$ replicas are faulty.

Properties:

- deterministic
- replicated
- with state
- with read/write operations
- *safety:* behave like a centralized service
- *liveness:* clients receive replies

Working on asynchronous Internet like environment!

”Repeat and Compare”

- content integrity
- untrusted nodes
- dynamic content generation

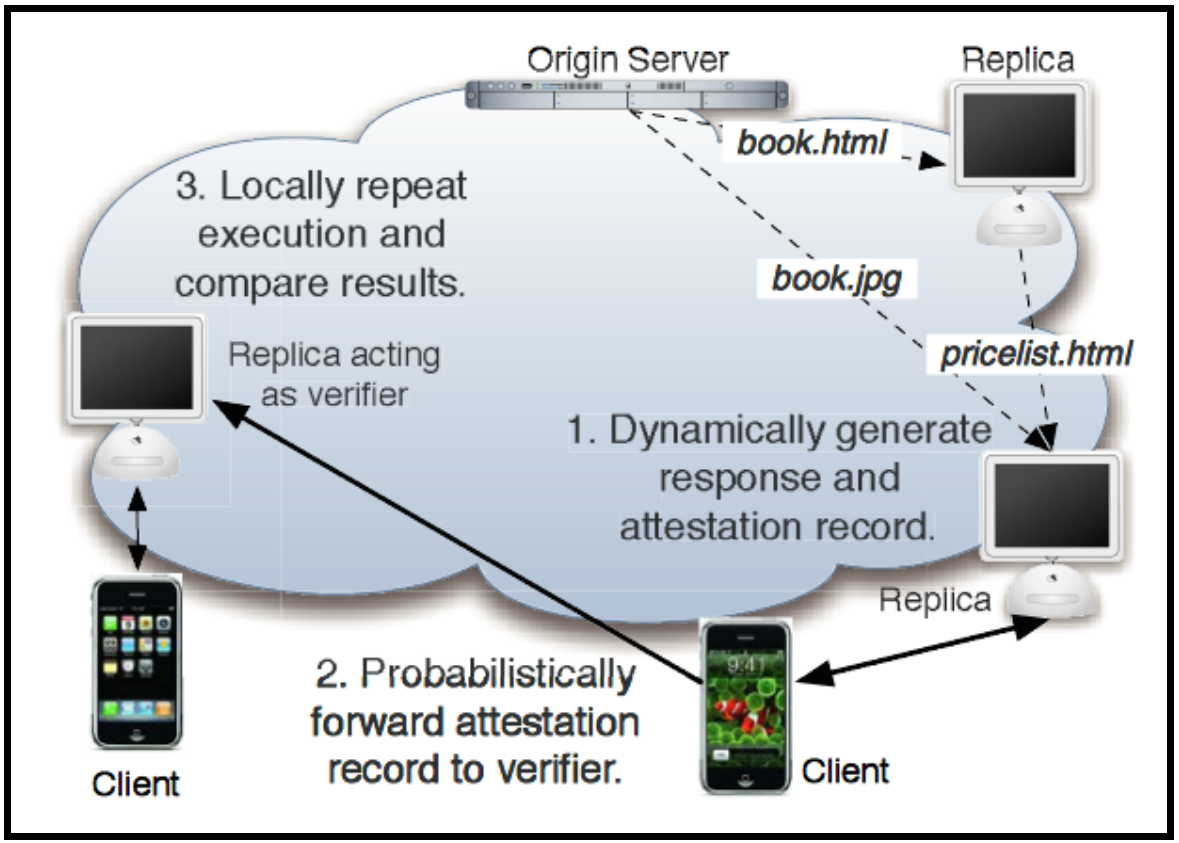
Peer-to-peer substrate:

- *repeat* content generation
- *compare* the results to detect misbehavior

Build on top of Na Kika.

Components:

- origin servers (trusted)
- supplier replicas (untrusted)
- verifier replicas
- clients



Requirements:

1. users can donate resources to the infrastructure
2. type of dynamic content should be restricted as little as possible
3. ensuring integrity must not rely on the ends (clients or servers)

Main challenges:

1. monitor responses sent to clients
2. repeat response generation
3. isolate misbehaving nodes

Attestation records:

Cryptographically bind a content supplier and its output.

- Origin servers -> content
- Replicas -> dynamically generated output

Clients can forward a small sample record to replicas verifiers.

Overhead:

Attestation records production add some overhead.

Content Producer Overhead: it signs each attestation record with its private RSA key -> significant overhead.

Verifier Overhead: it has to completely reprocess computations so overhead can easily be 100% than content generation.

Verifiers:

1. Locally *repeats* execution
2. Checks if the result matches the record sent by the client
3. When discover a bad replica it updates his list of suspected replicas

How to spot malicious verifiers?

Untrusted verifiers are monitored by a small **core of trusted verifiers**.

Content Integrity

Optimistic approach: clients accepts the response but informs the CDN about it.

Detection rather than prevention:

- with just one well-behaved replica is possible to detect bad ones
- detection is cheap (decoupled from request-response process)
- good enough
- discourage misbehaviour

Detection completeness:

misbeahving nodes will be suspected by the system.

Detection accuracy:

well-behaved nodes will not be suspected by the system.

Detection approaches:

Approach	Key Idea	Benefits and Drawbacks
Detection by Clients	Clients verify the integrity and freshness of the data they receive.	Protects static content. Resource-limited clients may not be able to verify integrity of dynamically generated content.
Detection by Spies	A set of spies imitates clients by sending requests and then verifying responses.	No changes to clients or servers, but cannot know if a spy is compromised.
Detection by Informers & Reputation	Verifiers use responses forwarded by clients to compute a replica's trust score.	Relies only on volunteers, but suffers from the "he said, she said" problem.
Detection by Trusted Verifiers using Attestation	Replicas held accountable through attestations. Response verification is performed by a small set of trusted verifiers.	Solves the "he said, she said problem". Does not scale.
Servers as Verifiers	Origin servers verify a sample of the responses using records from informers.	No need for globally trusted verifiers. Overloads popular servers. No ownership for aggregated content.
Detection by Untrusted and Trusted Verifiers	Relies on untrusted verifiers to assist trusted ones in verification.	Scales verification in the common case of correct responses by reducing the probability of detection.
Decentralized Trust	Verifiers keep local lists of misbehaving replicas. Clients learn about replicas using offline trust relationships.	No need for globally trusted verifiers. Detection is slower.

Decentralized Trust:

- Verifiers keep local lists of misbehaving replicas
- Trusted verifiers sends to the clients lists of unsuspected replicas

Completeness: each misbehaving replica must be detected by all correct verifiers -> overhead is on the order of the number of replicas in the system.

It's slow!

Untrusted and Trusted Verifiers:

Rely on untrusted verifiers: good scaling properties.

- as many verifiers as replicas in the CDN
- single invalid record is a proof of misbehaviour
- trusted verifiers process only the order of misbehaving replicas

Only probabilistic **completeness:** clients only forward an attestation record with probability p (to save bw).

So with Untrusted and Trusted Verifiers we got:

- attestation records -> detection **accuracy** guarantees
- sampled forwarding approach -> probabilistic **completeness**

Decentralized approach is slower!

Repeat:

To perform a *Repeat*, a replica needs to set up an equivalent environment to the original:

- external inputs (client request, original content)
- configuration parameters (server config., library versions)
- code to repeat

Identified by:

1. name
2. timestamp
3. value

Compare:

Composed by two stages:

1. Forwarding attestation records to verifiers
2. Detecting misbeahving replicas

Compare 1/2:

Forwarding attestation records to verifiers:

1. client checks attestation record consistency
2. forward record with probability $p \leq 1$ to a random verifier

Compare 2/2:

Detecting misbeahving replicas:

- **Centralized** detection:
  1. untrusted verifiers receive records from clients
  2. perform *Repeat*
  3. in case of misbehaving replica forward the record to a trusted verifier
- **Decentralized** detection:
  1. clients forward attestation records
  2. verifiers locally repeats executions
  3. a detected misbehaviour change in negative the related voting table
  4. trusted verifiers (*friends*) which sends to the clients lists of unsuspected replicas

Detection:

Assumptions:

- no new nodes enter or leave
- clients forward records with probability *p*
- *f* misbehaving replicas
- *g* well-behaved replicas (constant over time)

Probability for a client to receive corrupt content:

$$f/(f + g)$$

Detection properties:

1. P_d of detecting a misbehaving replica
2. T it takes to detect all misbeahving replicas

1. $\mathbf{P_d}$ misbehaving replica is detected:

Depends on:

- b number of incriminating records sent
- client forwarding to good verifier P_v

$$P_v = \frac{pg}{f+g}$$

$$P_d = 1 - \left(1 - \frac{pg}{f+g}\right)^b$$

If f is small then P_d improves as p increases; if f if large it masks effects of p (even if $p = 1, P_d$ can reach 1 only increasing b).

2. $\mathbf{T}$ detect all misbeahving replicas:

Let's assume:

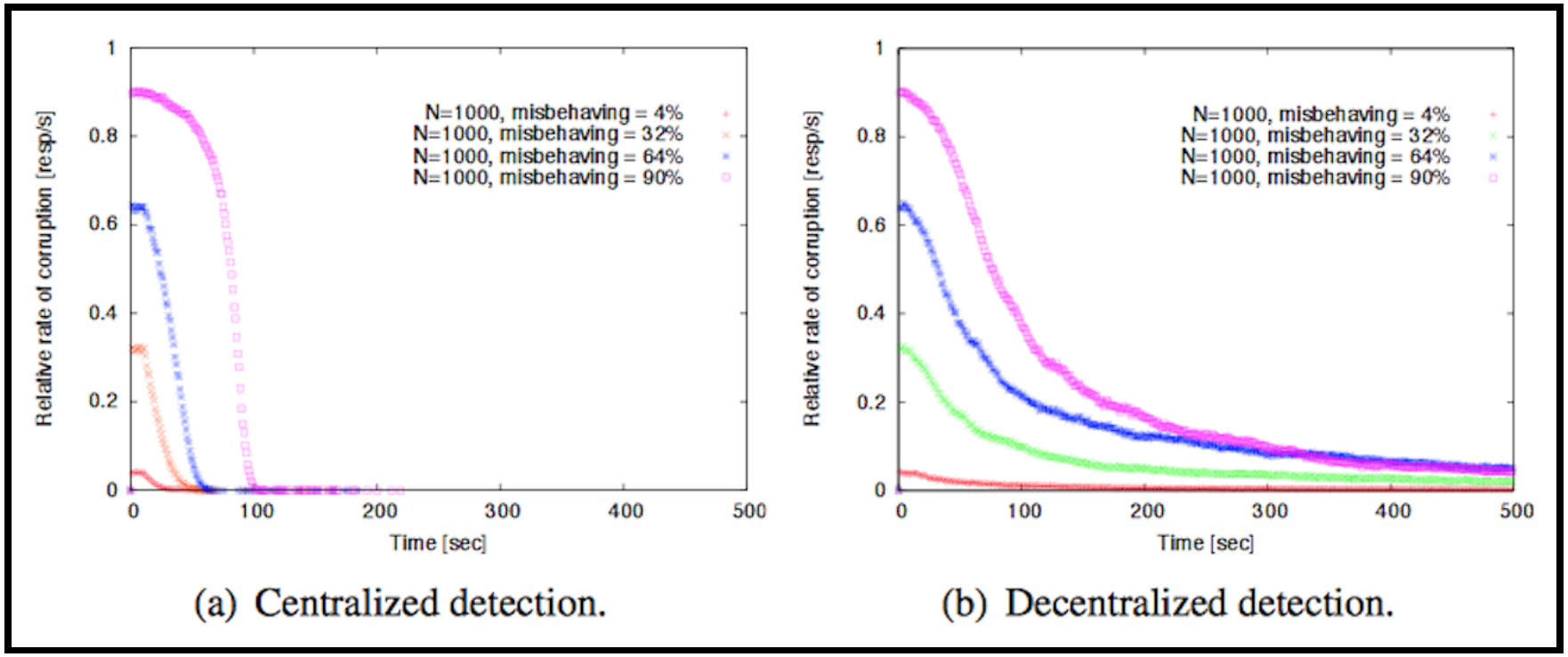
1. no network or computational latency -> lower bound estimate
2. there are F misbeahving replicas initially, no new replicas enter the system

$$T = \frac{F + g + g \ln F}{pg}$$

For large F it depends on how large is F compared to pg, for small F detection is faster for higher values of p.

Simulations:

Centralized vs decentralized, p = 0.1



Using a small set of trusted verifiers (size=4), misbeahving replicas are detected much faster than in the decentralized detection model.

Known issues:

1. Fault detection requires repeatable computations -> limited to determinism
2. Changing small number of high-value responses -> priority value in attestation records so high-value records have forward probability $p = 1$
3. No databases support

DoS: misbehaving nodes are motivated to serve in order to maximize damage -> not an issue.

?

References:

1. N. Michalakis R. Soule, R. Grimm, "Ensuring Content Integrity for Untrusted Peer-to-Peer Content Distribution Networks", 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 2007)
2. B.C Popescu, B. Crispo, A. Tanenanbaum "Secure data Replication over Untrusted Hosts", Proceedings of the 5th Usenix UNIX Security Symposium, Vol. 5, Salt Lake City, Utah, USA, June 1995, pp. 199-208
3. M. Castro, B. Liskov, "Practical Byzantine Fault Tolerance", Proceedings of the 3rd Symposium on Operating Systems Design and Implementation OSDI 1999
4. http://en.wikipedia.org/wiki/Content_delivery_network