Dependability and Security with Clouds-of-Clouds

lessons learned from \( n \) years of research

Miguel Correia

WORKSHOP ON DEPENDABILITY AND INTEROPERABILITY IN HETEROGENEOUS CLOUDS (DIHC13)
August 27th 2013, Aachen, Germany

Outline

- Motivation
- Opportunities and challenges
- Storage – DepSky
- Processing – BFT MapReduce
- Services – EBAWA
- Conclusions
Outline

Motivation
• Opportunities and challenges
• Storage – DepSky
• Processing – BFT MapReduce
• Services – EBAWA
• Conclusions

Clouds are complex so they fail
Cloud-of-Clouds

- Consumer runs service on a set of clouds forming a virtual cloud, what we call a cloud-of-clouds
- Related to the notion of federation of clouds
  - “Federation of clouds” suggests a virtual cloud created by providers
  - “Cloud-of-clouds” suggests a virtual cloud created by consumers, possibly for improving dep&sec

Cloud-of-Clouds dependability+security

- There is cloud redundancy and diversity
- so even if some clouds fail a cloud-of-clouds that implements replication can still guarantee:
  - Availability – if some stop, the others are still there
  - Integrity – they can vote which data is correct
  - Disaster-tolerance – clouds can be geographically far
  - No vendor lock-in – several clouds anyway
Outline

• Motivation

Opportunities and challenges

• Storage – DepSky
• Processing – BFT MapReduce
• Services – EBAWA
• Conclusions

Replication / geo-replication in clouds

• Provides opportunities and challenges
• Some data from Amazon EC2
  – Not different clouds but close enough
  – Data collected ~hourly during August 2-15, 2013
  – One micro instance (virtual server) per Amazon region
Geographical redundancy and diversity
Amazon EC2 regions and availability zones

- Each region is completely independent
- Each availability zone (AZ) is isolated
- Note: personal map, positions may not be accurate

Network redundancy and diversity

- ASs provide another level of diversity (most ISPs have more than one)
- ISPs observed on the August 2nd (a few changes were observed in 2 weeks)
- This is not the complete graph, several edges are missing
Latency: high and variant

Throughput: low and variant

- Same pairs as in previous slide but opposite order
- Important: the throughput is higher with better instances (we used micro)
Economic cost (data transfer)

- Cost for data transfer IN to EC2 from Internet: 0 $
- Cost for data transfer OUT from EC2 to Internet:
  - Vertical axis is data transferred and has logarithmic scale


CAP theorem

- It is impossible for a web service to provide the following three guarantees:
  - Consistency
  - Availability
  - Partition-tolerance
- Network diversity suggests partitions are unlikely
  - Nodes may get isolated but not sets of nodes from others
  - But relaxed consistency may be offered if they happen
  - Current research topic; we won’t address it
Outline

- Motivation
- Opportunities and challenges

Storage – DepSky

- Processing – BFT MapReduce
- Services – EBAWA
- Conclusions

DepSky

- (Client-side) library for cloud-of-clouds storage
  - File storage, similar to Amazon S3: read/write data, etc.
- Use storage clouds as they are:
  - No specific code in the cloud
- Data is updatable
  - Byzantine quorum replication protocols for consistency
Write protocol

Cloud A → WRITE FILE
Cloud B → WRITE FILE
Cloud C → WRITE FILE
Cloud D → WRITE FILE

ACK

ACK

Read protocol

Cloud A → REQUEST FILE
Cloud B → REQUEST FILE
Cloud C → REQUEST FILE
Cloud D → REQUEST FILE

REQUEST METADATA

REQUEST METADATA

REQUEST METADATA

highest version number (+fastest or cheapest cloud)

FILE

File is fetched from other clouds if signature doesn’t match the file
Limitations of the solution so far

- Data is accessible by cloud providers
- Requires $n \times |\text{Data}|$ storage space

Combining erasure codes and secret sharing

Only for data, not metadata

Data

Cloud A
Cloud B
Cloud C
Cloud D

Data
Data
Data
Data

F1 S1 F2 S2 F3 S3 F4 S4

Cloud A Cloud B Cloud C Cloud D

K

K

key

encrypt

disperse

share

Encrypted so data can’t be read at a cloud!
Only twice the size of storage, not 4 times!
DepSky latency
100KB files, clients in PlanetLab nodes

DepSky read latency is close to the cloud with the best latency

DepSky write latency is close to the cloud with the worst latency

Lessons from Depsky

• Provides: availability, integrity, disaster-tolerance, no vendor lock-in, confidentiality

• Insights:
  – Some clouds can be faulty so we need Byzantine quorum system protocols (to reason about subsets of clouds)
  – Signed data allows reading from a single cloud, so faster or cheaper than average
  – Erasure codes can reduce the size of data stored
  – Secret sharing can be used to store cryptographic keys in clouds (avoiding the need of a key distribution service)
Outline

• Motivation
• Opportunities and challenges
• Storage – DepSky

Processing – BFT MapReduce

• Services – EBAWA
• Conclusions

What is MapReduce?

• Programming model + execution environment
  – Introduced by Google in 2004
  – Used for processing large data sets in clusters of servers
• Hadoop MapReduce, an open-source MapReduce
  – The most used, the one we have been using
  – Includes HDFS, a file system for large files
MapReduce basic idea

Job submission and execution
The problem

• The original Hadoop MR tolerates the most common faults
  – Job tracker detects and recovers crashed/stalled map/reduce tasks
  – Detects corrupted files (a hash is stored with each block)

• But execution can be corrupted, tasks can return wrong output

• and clouds can suffer outages

BFT MapReduce

• Basic idea: to replicate tasks in different clouds and vote the results returned by the replicas
  – Inputs initially stored in all clouds
Original MR – Map perspective

BFT MR – Map perspective
Original MR – Reduce perspective

BFT MR – Reduce perspective
Deferred execution

• Faults are uncommon; consider max. of \( f \) faults
• JT creates only \( f+1 \) replicas in \( f+1 \) clouds (\( f \) in standby)
• If results differ or one cloud stops, request 1 more (up to \( f \))

Distributed job tracker

• Job tracker controls all task executions in the task trackers (e.g., start task, detect faults)
  – If job tracker is in one cloud, separated from many task trackers by the internet
    • high latency to control operations
    • single point of failure
• Distributed job tracker
  – Each cloud has one job tracker (JT)
  – Each JT controls the tasks in its cloud, no “remote control”
All this communication through the WAN => high delay and $ cost
  - data transferred per pair can even be the size of the split (e.g., MBs)

Solution: digest communication

- Reduces fetch the map task outputs
  - Intra-cloud fetch: output fetched normally
  - Inter-cloud fetch: only hash of the output fetched
Makespan varying parallelism

- Estimated analytically based on another BFT-MR we implemented; $f=1$

Lessons from BFT MapReduce

- Provides: availability, integrity, disaster-tolerance, no vendor lock-in (no confidentiality)
- Insights:
  - Tasks can be replicated in different clouds to mask faulty executions / faulty clouds
  - Defer execution to reduce # tasks executed without faults
  - Control components should be placed in all clouds to avoid control operations between clouds (high delays)
  - Send only digests between clouds to avoid huge communication delays and costs
Outline

• Motivation
• Opportunities and challenges
• Storage – DepSky
• Processing – BFT MapReduce

Services – EBAWA
• Conclusions

State Machine Replication (SMR)

• Can be used to replicate “any” service
  – Ex: file sys., k.v. store, DBs, authentication serv., coordination serv.,…
  – All replicas start in the same state
  – All replicas execute the same requests in the same order

Fault-tolerant because operation does not depend on all replicas, $f$ can be faulty
BFT SMR is expensive in WANs

- Example: PBFT (Castro&Liskov’99)
  - Several communication steps, messages, votes
  - Ok for LANs but if steps are through a WAN...

```
Client
Replica 1
Replica 2
Replica 3
Replica 4
```

\[ f=1 \]

EBAWA
Efficient Byzantine Algorithm for Wide Area networks

- EBAWA is a BFT SMR algorithm like PBFT...
- ...but with a set of mechanisms for making it efficient in WANs...
- ...which make it adequate for clouds-of-clouds
Unique Sequential Identifier Generator service (USIG)

- Replicas include a trusted module: USIG
  - Local module, implemented to be trusted (e.g., in hardware), simple interface
  - Simple: monotonic counter + cryptographic mechanism
- Interface:
  - createUI: assigns a signed unique identifier to a message m
  - verifyUI: checks if the unique id is valid for message m

Benefits of USIG

- USIG prevents certain kinds of faults/misbehavior
  - Faulty replicas can’t send 2 messages with the same id
- This allows cutting:
  - The number of servers from 3f+1 to 2f+1
  - Number of communication steps by one (lower latency)
- Together they greatly reduce the #messages:
Rotating primary

- The primary only orders a batch of requests per view, then the next replica becomes the primary
  - Prevents performance attacks (e.g., faulty server slows down service) – critical in WANs due to high timeouts
  - Reduces latency as client can access the closest replica
  - Provides load balancing

Asynchronous views

- A replica starts an agreement as soon as it receives a client request by sending a prepare message
  - Servers without pending client requests skip their turn by sending a special message
Performance in PlanetLab – Europe

• Nodes:
  – Portugal (client)
  – France
  – Italy
  – Germany (primary)
  – Spain
• EBAWA avg. latency
  43% lower than JPBFT’s

Lessons from EBAWA

• Provides: availability, integrity, disaster-tolerance, no vendor lock-in (no confidentiality)
• Insights:
  – Reducing the communication steps (with the USIG) reduces the latency
  – Reducing the number of replicas (with the USIG) reduces costs
  – Rotating the primary allows preventing performance attacks, load balancing, client can access closest replica (reduc. latency)
  – Asynchronous views reduce waiting, thus latency
  – Waiting for n-f clouds allows disregarding the f with higher RTT
Outline

- Motivation
- Opportunities and challenges
- Storage – DepSky
- Processing – BFT MapReduce
- Services – EBAWA

Conclusions

Lessons learned

- Clouds-of-clouds: solution for consumers to create dependable & secure clouds on top of cloud offerings
  - We’ve seen clouds-of-clouds for: storage, processing, services
- Usable or latency/cost too high?
  - Latency: if we disregard processing delays, the latency is a few RTTs, but the same with “normal” clouds (e.g., min 2 RTTs for an HTTP request)
  - Cost: higher, but dependability & security aren’t free
Lessons learned

- **Important design goals:**
  - to reduce the number of communication steps
  - to reduce the data sent out of the individual clouds
  - to reduce the number of messages
  - to reduce the size of the data stored
  - to reduce the number of replicas
  - to do control locally in every cloud

- **We’ve seen several mechanisms to tackle these goals:**
  - Byzantine quorum system protocols; auto-verifiable (signed) files; erasure codes; task replication; deferred task execution; local control components; digest communication between clouds; the USIG service; rotating primary; asynchronous views; waiting for $n-f$ replicas

Thank you!

Further reading:
- My paper at DIHC 2013’s post-proceedings