

Multi-Cloud Computing

Motivation

Redundant Array of Cloud Storage (RACS)

„Vendor Lock-In“-Problem

- Typische Vorgehensweise bei der Migration eines Diensts in die Cloud
 - **Auswahl eines Cloud-Anbieters** anhand bestimmter Kriterien (Beispiele)
 - Kostenabschätzung
 - Leistungsgarantien (*Service Level Agreements*)
 - Anpassung des Diensts auf die spezifischen Gegebenheiten einer Cloud
 - Probleme
 - Preismodell, Leistungsgarantien,... eines Anbieters können sich ändern
 - Ein Anbieterwechsel ist in der Regel sehr aufwendig
- Mögliche **Gründe für erschwerten Anbieterwechsel** (Beispiele)
 - Technisch nicht kompatible Cloud-Systeme
 - Heterogene Schnittstellen
 - Anbieterspezifische Funktionalität
 - Hoher zeitlicher/finanzieller Aufwand [z. B. falls Transfer großer Datenmengen erforderlich.]
- Herausforderungen
 - Wie lässt sich der **Aufwand für einen Anbieterwechsel reduzieren**?
 - Wie kann ein Dienst auf Basis mehrerer Clouds bereitgestellt werden?

Redundant Array of Cloud Storage (RACS)

- Naiver Ansatz für redundante Speicherung von Daten
 - **Vollständige Replikation der Daten** über mehrere Clouds
 - Nachteile
 - Hohe Kosten für Datenspeicherung und -transfer
 - Hinzufügen eines weiteren Anbieters erfordert Umkopieren aller Daten
- **Redundant Array of Cloud Storage (RACS)**
 - Anwendungsszenario
 - Fokus auf Anbieterwechsel aus wirtschaftlichen Gründen
 - Wechsel kündigt sich vergleichsweise lange im Voraus an
 - Einbindung möglichst vieler Anbieter sinnvoll
 - Vorbild: Redundant Array of Independent Disks (RAID)
 - Redundante Speicherung von Daten mittels **Erasure-Codes**
- Literatur
 - Hussam Abu-Libdeh, Lonnie Princehouse, Hakim Weatherspoon
RACS: A case for cloud storage diversity
Proceedings of the 1st Symposium on Cloud Computing (SoCC '10), S. 229–240, 2010.

Erasure-Codes

Einschub

- Allgemeines
 - **Verfahren zur Vorwärtsfehlerkorrektur**
 - Einsatzbereiche (Beispiele): Datenübertragung, -speicherung
 - Beispiel *Reed-Solomon-Codes*: RAID 6, CDs,...
- Grundlegender Ansatz
 - Zerlegung eines zu speichernden Objekts O in m **gleichgroße Teile**
 - Berechnung von $n > m$ Fragmenten basierend auf den m Teilen
 - **Besondere Eigenschaft des Abbildungsverfahrens**: Die originalen m Teile lassen sich aus m beliebigen der n Fragmente rekonstruieren
 - Kein Datenverlust solange höchstens $n - m$ Fragmente verloren gehen
- **Speicherbedarf** abhängig von m und n
 - Faktor $x = \frac{n}{m}$ im Vergleich zur nicht-redundanten Speicherung
 - Beispiel: Verteilung auf 7 Anbieter, Tolerierung eines Weg- bzw. Ausfalls
 - $m = 6, n = 7 \Rightarrow x \approx 1.17$
 - Vergleich: Bei naivem Ansatz $x = 7$ (7 Anbieter) bzw. $x = 2$ (Tolerierung eines Ausfalls)

Schnittstelle

■ Datenmodell

- Anlehnung an das **Datenmodell von Amazon S3**
- Verwaltung von Schlüssel-Wert-Paaren
- Datenstrukturen
 - Verzeichnisse: *Buckets*
 - Binärdateien: *Objekte*

■ Schnittstelle

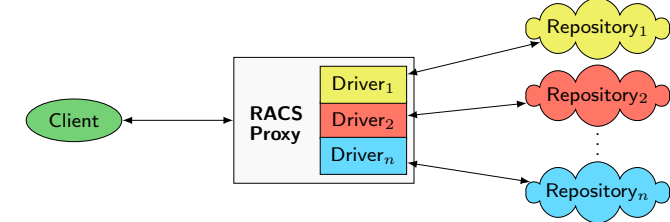
- Ziel: Generische, vom einzelnen Anbieter unabhängige Schnittstelle
- Verfügbare Operationen

Operation	Beschreibung
<code>create()</code>	Anlegen eines Bucket
<code>put()</code>	Hinzufügen eines Objekts zu einem Bucket
<code>get()</code>	Auslesen eines Objekts
<code>delete()</code>	Löschen eines Objekts bzw. Bucket
<code>list()</code>	Auslesen von Metadaten eines Bucket

Architektur

■ Basisarchitektur

- Client-Anwendung [z. B. unmodifizierter Amazon-S3-Client]
- **RACS-Proxy**
 - Übersetzung von Client-Aufrufen in Anfragen an verschiedene Clouds
 - Spezifischer Adapter für jeden Cloud-Speicher-Anbieter
- Repositories: **Datenspeicher bei unterschiedlichen Cloud-Anbietern**



■ Verteiltes RACS

- Erweiterung der Basisarchitektur um weitere RACS-Proxies
- **Koordinierung zwischen RACS-Proxies** mittels ZooKeeper

Funktionsweise

■ Bearbeitung von Anfragen

- `put()`-Anfragen
 - Berechnung von n Objektfragmenten mittels Erasure-Codes
 - Verteilung der Objektfragmente auf n Cloud-Speicher-Anbieter
 - Vollständige Replikation der Metadaten auf alle n Repositories
- `get()`-Anfragen
 - Holen von m Objektfragmenten aus verschiedenen Repositories
 - Rekonstruktion des angeforderten Objekts
- `list()`-Anfragen
 - Auslesen der Metadaten aus einem beliebigen Repository
 - Vollständige Replikation von Metadaten → einzelne Operation reicht aus

■ Optimierte Auswahl von Repositories mittels *Policy Hints*

- Vorgaben, welche Repositories (z. B. bei Leseanfragen) zu bevorzugen sind
- Beispiele für Kriterien
 - Kosten für Datentransfer
 - Geografisch bedingte Latenzen