Überblick

Infrastructure as a Service

- Bereitstellung von Hardware-nahen Ressourcen
  - Virtuelle Maschinen auf Systemebene
  - Zuverlässiger und hochverfügbarer Datenspeicher
- Angestrebtes Preismodell: *Pay-as-you-go*
  - Abrechnung nach tatsächlich verbrauchten Ressourcen
  - Keine Fixkosten für Anschaffung, Betrieb und Wartung

  → Kosten orientieren sich im Optimalfall am Gewinn
- Dynamische Skalierbarkeit in beide Richtungen
  - Hinzufügen weiterer virtueller Maschinen bei Bedarfsspitzen
  - Herunterfahren von virtuellen Maschinen bei zu geringer Auslastung
- Literatur

Peter Sempolinski and Douglas Thain
A comparison and critique of Eucalyptus, OpenNebula and Nimbus
Proceedings of the 2nd International Conference on Cloud Computing Technology and Science (CloudCom '10), pages 417–426, 2010.

IaaS-Beispiel: Amazon EC2

- Virtuelle Maschinen verschiedener Größe (Beispiele)
  - *Micro*-Instanz: 613 MB Speicher, bis zu 2 ECUs
  - *Cluster Compute Eight Extra Large*-Instanz: 60,5 GB Speicher, 88 ECUs

  [Hinweis: „Eine EC2 Compute Unit (ECU) bietet die entsprechende CPU-Kapazität eines 1,0- bis 1,2-GHz-Opteron- oder -Xeon-Prozessors von 2007.“, Quelle: `http://aws.amazon.com/de/ec2/`]
- Virtuelle Maschinen verschiedener Kategorien
  - *On-Demand Instance*
    - Sofortige Verfügbarkeit
    - Abrechnung nach angefangener Stunde
  - *Reserved Instance*
    - Grundpreis für längeren Zeitraum, vergünstigter Stundenpreis
    - Unterkategorien für verschiedene Auslastungsstufen
  - *Spot Instance*
    - Abhängig von der aktuellen Cloud-Auslastung wird ein Referenzpreis definiert
    - Ausführung der VM sobald Referenzpreis unter vordefinierten Wert fällt
- Nähere Informationen: `http://aws.amazon.com/de/ec2/`

Herausforderungen

- Kompromiss bei der Platzierung von virtuellen Maschinen
  - Viele virtuelle Maschinen auf dem selben Rechner $\rightarrow$ **hohe Auslastung**
  - Möglichst gleichmäßige Aufteilung der virtuellen Maschinen auf die vorhandenen Rechner $\rightarrow$ **geringe Beeinflussung** der VMs untereinander
  - In geringer geographischer Nähe zum Nutzer $\rightarrow$ **niedrige Latenz**
  - Geographisch weit entfernt von anderen virtuellen Maschinen der selben Anwendung $\rightarrow$ **hohe Ausfallsicherheit** des gesamten Dienstes
- Im Folgenden betrachtet
  - Wie kann eine weltumspannende Cloud-Infrastruktur mit Datenzentren auf verschiedenen Kontinenten realisiert werden?
  - Wie lässt sich die durch Virtualisierung erzielte Isolation auf Netzwerke von virtuellen Maschinen ausdehnen?
  - Wodurch kann Datenverlust bei Katastrophen vorgebeugt werden?
  - Wie lässt sich Lastbalancierung zwischen Speicherknoten erzielen?

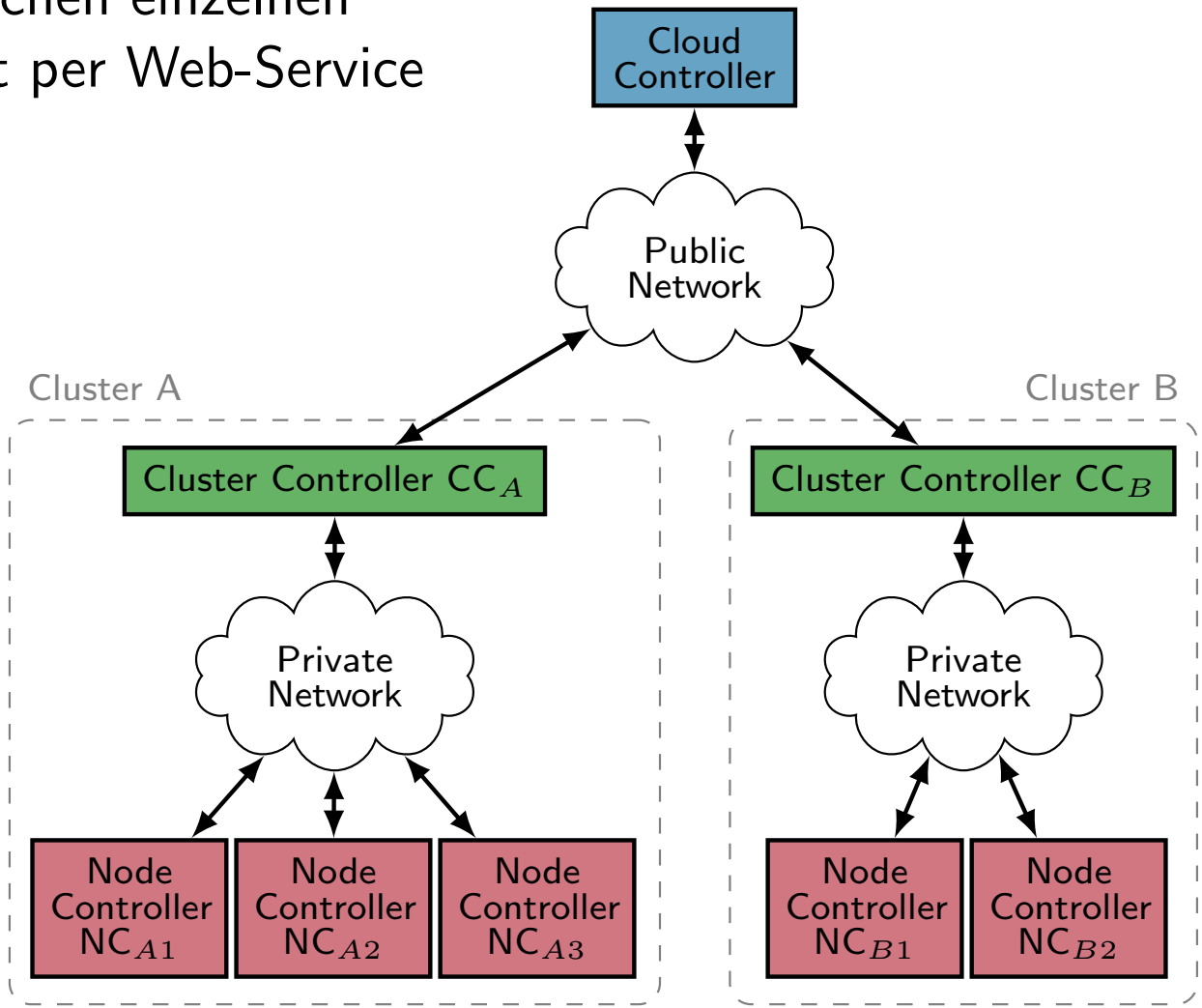
Eucalyptus

- Motivation
  - Einsatz von proprietären Implementierungen in kommerziellen IaaS-Clouds
  - Kaum Informationen über den Aufbau solcher Systeme vorhanden
  - Beschränkte Zugangsmöglichkeiten für Forscher
- Eucalyptus
  - Framework zur Verwaltung privater bzw. hybrider Clouds
  - Zielgruppe: Universitäten und kleinere Firmen
  - Anlehnung an Amazon EC2 bzw. Amazon S3
    - `eucatools` für Interaktion mit dem Framework
    - Client-Schnittstelle für Datenspeichersystem
- Literatur

Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli et al.
The Eucalyptus open-source cloud-computing system
Proceedings of the 9th International Symposium on Cluster Computing and the Grid (CCGrid '09), pages 124–131, 2009.

Architektur

- Kommunikation zwischen einzelnen Komponenten erfolgt per Web-Service
- Controller-Hierarchie
  - Cloud
  - Cluster
  - Node
- *Walrus*
  - Datenspeicher
  - Archiv für Images virtueller Maschinen
  - Zugriff von inner- und außerhalb der Cloud möglich



Komponenten

- *Cloud Controller*
  - Zuständigkeitsbereich: komplette Eucalyptus-Cloud
  - Schnittstelle zum Cloud-Nutzer bzw. -Administrator
    - Authentifizierung von Nutzern
    - Verwaltung von virtuellen Maschinen
- *Cluster Controller*
  - Zuständigkeitsbereich: Gruppe von Rechnern [Ausgangspunkt: Region in Amazon EC2]
  - Bearbeitung von Anfragen des Cloud Controller
    - Auswahl der Node Controller für den Start einer virtuellen Maschine
    - Analyse der Kapazitäten für bestimmte VM-Typen
  - Konfiguration und Überwachung von *Virtual Network Overlays (VNOs)*
- *Node Controller*
  - Zuständigkeitsbereich: (einzelner) lokaler Rechner
  - Bearbeitung von Anfragen des zugehörigen Cluster Controller
    - Starten und Stoppen virtueller Maschinen
    - Berichte über Zustände lokaler virtueller Maschinen
    - Übersicht über Ressourcen (z. B. Anzahl an CPUs, freier Festplattenspeicher)

Start einer virtuellen Maschine

- Cloud Controller
  - Empfang einer Anfrage vom Nutzer
  - Überprüfung der Verfügbarkeit von Ressourcen
  - Senden einer *Ressourcenreservierung* an entsprechenden Cluster Controller
  - Abwarten einer Bestätigung, dass die Ressourcen reserviert wurden
  - Senden einer Anweisung an den Cluster Controller die VM zu starten
- Cluster Controller
  - Auswahl des Rechners, auf dem die VM gestartet werden soll
  - Anwendung der *First-Fit*-Strategie
- Node Controller
  - Bereitstellung des VM-Image auf dem Zielrechner (Varianten)
    - Transfer aus dem Image-Archiv von Walrus
    - Verfügbarkeit im lokalen Image-Cache
  - Konfiguration des Netzwerks (zusammen mit dem Cluster Controller)
  - Anweisung an den VMM das VM-Image zu booten

→ Nutzer kann per `ssh` auf seine virtuelle Maschine zugreifen

Virtual Network Overlays

- Anforderungen
  - **Isolation**: Eine VM eines Nutzers muss mit anderen VMs des selben Nutzers kommunizieren können, jedoch nicht mit VMs anderer Nutzer
  - **Erreichbarkeit**: Mindestens eine virtuelle Maschine jedes Nutzers muss von außerhalb der Cloud erreichbar sein
- Umsetzung mittels *Virtual Network Overlays*
  - Realisierung der Isolation
    - Einrichtung eines separaten VLANs für jeden Nutzer
    - Jedes VLAN verwendet ein eigenes IP-Subnetz
    - Cluster Controller
      - Bei Bedarf: Routing zwischen IP-Subnetzen
      - Isolation durch Firewall-Regeln
    - TCP- oder UDP-Tunnel zwischen verschiedenen Clustern
  - Einfluss auf Erreichbarkeit
    - Verwendung privater IP-Adressen $\rightarrow$ VMs von außen nicht zugänglich
    - Bei Bedarf Adressumsetzung von öffentlichen auf private IP-Adressen

Walrus

- Schnittstelle: Orientierung an Amazon S3
  - Zugriff per REST oder SOAP
  - Verwaltung von Daten in *Objekten* und *Buckets*
  - Verwendung von Amazon-S3-Tools möglich
- Archiv für VM-Images virtueller Maschinen
  - Bestandteile eines VM-Image
    - Root Filesystem
    - Kernel Image
    - Ramdisk Image
  - Ablegen eines VM-Image
    - Komprimierung, Verschlüsselung und Aufteilung
    - Beschreibung des Image in einem *Manifest*
  - Bearbeitung von Node-Controller-Anfragen
    - Zusammenfügen, Verifizieren, Entschlüsselung des VM-Image
    - Transfer des Image zum Node Controller
  - Cache für entschlüsselte VM-Images

Windows Azure Storage

- Anforderungen
  - Starke Konsistenz
  - Globaler Namensraum
  - Kein Datenverlust bei Katastrophen
  - Niedrige Kosten
- Windows Azure Storage
  - Einheitliches Speichersystem für unterschiedliche Nutzdaten
  - Trennung des Datenspeichers vom Rest der Cloud
  - Rückgriff auf das Domain Name System (DNS)
  - Georeplikation über mehrere Datenzentren
- Literatur

Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan et al. **Windows Azure Storage: A highly available cloud storage service with strong consistency.** *Proceedings of the 23rd Symposium on Operating Systems Principles (SOSP '11)*, pages 143–157, 2011.

Adressierung von Datenobjekten

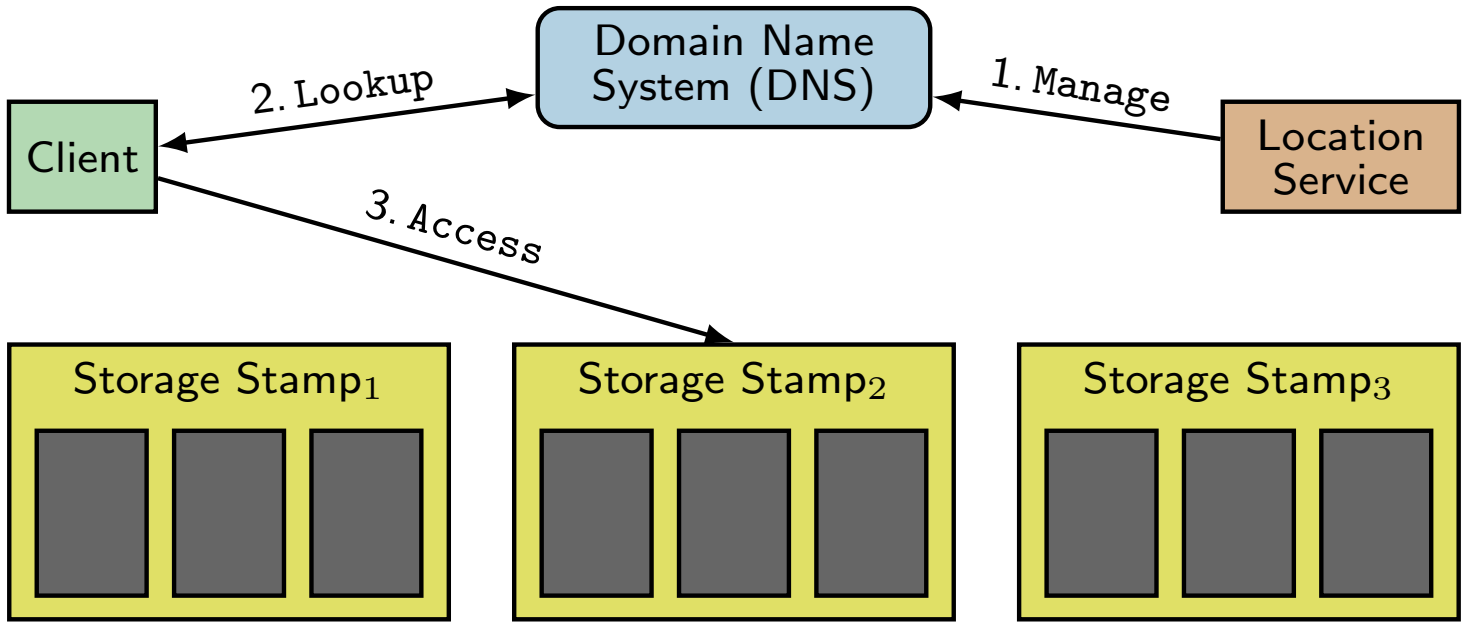
- Verfügbare Datenobjekte
  - *Blobs*
  - Tabellen
  - Warteschlangen
- Typischer Einsatz von Objekten
  - Eingabedaten: Blobs
  - Zwischenergebnisse und Ausgabedaten: Blobs oder Tabellen
  - Koordinierung: Warteschlangen
- Globaler partitionierter Namensraum

[Protokoll]`://`[Konto]`.`[Dienst]`.core.windows.net/`[Partition]`/`[Objekt]

  - Protokoll: `http` bzw. `https`
  - Kontoname des Nutzers ($\rightarrow$ Speicherort) als Teil des DNS-Host-Namens
  - Dienst: `blob`, `table` oder `queue`
  - Identifikation eines Objekts mittels Partitions- und Objektname

Architektur

- *Storage Stamp*
  - Gruppe aus mehreren Racks mit insgesamt 2 PB Speicher (künftig 30 PB)
  - Racks besitzen eigene Netzwerk- und Stromanbindungen → *Fehlerdomänen*
  - Stamp von außen über eine virtuelle IP-Adresse erreichbar
- Ortsdienst (*Location Service*)
  - Zuordnung von Nutzerkonten zu Stamps
  - Stamp-Auswahl für neue Konten
  - Aktualisierung der Stamp-Adressen im DNS



Storage Stamp

- *Front-End Layer*
  - Authentifizierung eintreffender Anfragen
  - Weiterleitung von Anfragen an den Partition Layer
- *Partition Layer*
  - Verwaltung von Blobs, Tabellen und Warteschlangen
    - Zusammenfassung kleiner Objekte
    - Aufteilung großer Objekte in Partitionen
  - Verwaltung von Partitionen
    - Einteilung und Zuordnung zu Servern
    - Lastverteilung zwischen Servern
  - Replikation über mehrere Stamps
- *Stream Layer* [→ Siehe nächste Vorlesung.]
  - Direkter Zugriff auf Festplatten
  - Bereitstellung von *Datenströmen (Streams)*
  - Stamp-interne Replikation

Replikation

- Replikation innerhalb eines Stamp (*Intra-Stamp Replication*)
  - Aufgabe des Stream Layer
  - Synchrone Replikation während des Schreibvorgangs
  - Speicherung der Replikate in unterschiedlichen Fehlerdomänen
  - Replikation auf Binärdaten-Ebene
  - Im Fehlerfall: Wechsel der Replikatgruppe
- Replikation zwischen Stamps (*Inter-Stamp Replication*)
  - Aufgabenverteilung
    - Location Service: Nutzerkonto-spezifische Konfiguration
    - Partition Layer: Durchführung
  - Asynchrone Replikation im Hintergrund [Im Durchschnitt ca. 30s nach dem Schreibvorgang.]
  - Replikation auf Objektebene
  - Im Fehlerfall: Failover durch Anpassung des DNS-Eintrags eines Kontos

- Zentrale Datenstruktur: Objekttabelle (*Object Table*)
  - Speicherung sehr großer Datenmengen [→ Petabytes]
  - Aufteilung in disjunkte *RangePartitions*
  - Beispiele
    - *Account Table*: Verwaltung von Informationen über Nutzerkonten
    - *Blob Table*: Tabelle mit allen Blobs eines Stamp
    - *Partition Map Table*: Zuordnung von RangePartitions zu Objekttabellen
- Komponenten
  - *Lock Service*
    - Vergleiche: Chubby Lock Service [→ Siehe spätere Vorlesung.]
    - Vergabe von *Leases* für RangePartitions an Partition Server
  - *Partition Server*
    - Verwaltung der ihm zugeteilten RangePartitions
    - Persistente Speicherung von Daten mittels Stream Layer
  - *Partition Manager*
    - Zuweisung von RangePartitions zu Partition-Servern
    - Mehrere Instanzen pro Stamp: Auswahl eines Anführers per Lock Service

- Datenstrukturen pro RangePartition (Beispiele)
  - Persistente Datenstrukturen im Stream Layer
    - *Metadaten-Strom*: Verwaltungsinformationen [z. B. Verweise auf andere Ströme]
    - *Commit-Log-Strom*: Log für kürzlich durchgeführte Modifikationen
    - *Row-Data-Strom*: Speicherung von Checkpoints
  - Flüchtige Datenstrukturen im Arbeitsspeicher
    - *Memory Table*: Modifikationen, die noch nicht Teil eines Checkpoint sind
    - *Index Cache*: Cache für Checkpoint-Indizes
    - *Row Data Cache*: Cache für Checkpoint-Inhalte
- Vorgehensweise bei Schreibaufrufen
  1. Anhängen der Operation an den Commit-Log-Strom
  2. Aktualisierung der betroffenen Tabellenzeile in der Memory Table
- Erstellung von Checkpoints
  1. Auslöser: Commit Log / Memory Table erreichen eine bestimmte Größe
  2. Erzeugung eines Checkpoint aus dem Inhalt der Memory Table
  3. Aufräumen des Commit Log

- Migration einer RangePartition von PS_A nach PS_B
  1. Der Partition Manager weist PS_A an, die Partition zu migrieren
  2. PS_A erstellt Checkpoint der Partition
  3. Partition Manager aktualisiert die Partition Map Table
  4. PS_B lädt RangePartition
- Teilung einer von PS_C verwalteten RangePartition P
  1. Der Partition Manager weist PS_C an, die Partition zu teilen
  2. PS_C erstellt Checkpoint von P
  3. PS_C erzeugt die Datenstrukturen für die Partitionsteile P_1 und P_2 basierend auf den Inhalten der Datenstrukturen von P
  4. PS_C verwaltet sowohl P_1 als auch P_2
  5. Partition Manager aktualisiert die Partition Map Table
  6. P_1 oder P_2 wird auf einen anderen Partition Server migriert
- Zusammenlegung zweier RangePartitions: Umkehrung zur Teilung

Zusammenfassung

- Infrastructure as a Service
  - Bereitstellung von Rechenkapazität und Datenspeicher
  - Dynamische Skalierbarkeit in beiden Richtungen
- Eucalyptus
  - Open-Source-Implementierung für private IaaS-Clouds
  - Schnittstellen: Orientierung an Amazon EC2
  - Hierarchischer Aufbau zur Unterstützung separater Cluster
- Windows Azure Storage
  - Datenspeichersystem der Microsoft-Cloud
  - Schichten
    - Front-End Layer
    - Partition Layer
    - Stream Layer
  - Replikation auf zwei Ebenen
    - Über verschiedene Fehlerdomänen innerhalb eines Storage Stamp hinweg
    - Georeplikation über mehrere Storage Stamps