Hadoop in der Cloud: Migration von Hunderten von Petabytes in die Cloud

Bogdan Lashkov, Edgar Linde
Innovaforge IT Consulting & Softwareentwicklung GmbH
Frankfurt am Main, Deutschland
{bogdan,edgar}@innovaforge.de

***Abstract*—Die Migration von Hardware in die Cloud scheint in den meisten Fällen nichts Schwieriges oder Überraschendes mehr zu sein - der Trend zur Bereitstellung von Lösungen in der Cloud ist weit verbreitet und gut etabliert. Aber während es einfach ist, kleine IT-Komponenten in die Cloud zu verlagern, sieht es bei globalen Systemen mit Hunderten von Petabytes an Daten etwas anders aus - solche Fälle sind selten.**

Stichwörter: Hadoop, HDFS, cloud migration, big data

Mein Name ist Bogdan Lashkov. Ich bin Geschäftsführer der Innovaforge IT Consulting & Softwareentwicklung GmbH und verantwortlich für die Kundenbetreuung und Beratung in den Bereichen Data Science, Big Data und Künstliche Intelligenz. In diesem Artikel werde ich über unsere Erfahrungen bei der Migration von Hadoop von Bare Metal in die Cloud berichten: wo wir angefangen haben, welche Optionen wir in Betracht gezogen haben, wie wir die Migration strukturiert haben und worauf wir dabei gestoßen sind.

I. Erstinstallation

Hadoop ist eine wichtige Komponente der IT-Infrastruktur eines unserer Kunden, die wir aktiv nutzen, um Daten aus verschiedenen Quellen zu verarbeiten.

Unser Hadoop ist ein ziemlich großes System:

- Die Clustergröße auf den Festplatten übersteigt 250 Petabyte;
- Arbeitsspeicherkapazität - 200 Terabyte, die auf 7 Rechenzentren verteilt sind;
- führt jeden Tag über 10.000 Aufgaben aus;
- Hat einen Zusammenschluss von drei Hauptclustern.

Es gibt zwei wichtige Teilsysteme in Hadoop:

- YARN;
- HDFS.

YARN ist ein Subsystem zur Ausführung von Berechnungen. Es ist auf dem Rechner, auf dem es eingesetzt wird, CPU- und RAM-intensiv. Es besteht aus:

- Ein Ressourcen-Manager, der Ressourcen zuweist;
- Nodemanager, auf dem ressourcenintensive Anwendungen laufen.

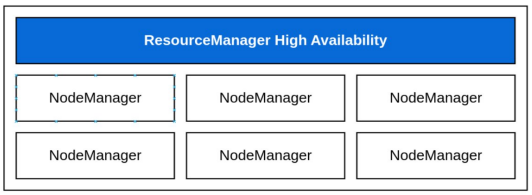

Bild 1 Ressourcenmanager Architektur

HDFS ist ein Speicher-Subsystem. Es besteht aus:

- namenode, wo die Blockinformationen gespeichert werden;
- datanode, in dem die Datenblöcke gespeichert sind.

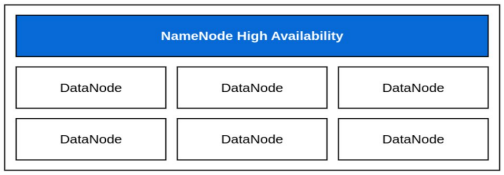

Bild 2 Namensknoten Architektur

Im Bereich "Hardware" sieht es folgendermaßen aus:

- Es gibt Master-Hosts, auf denen Namensknoten und Ressourcen-Manager bereitgestellt werden;
- Es gibt einen Worker-Host, auf dem Nodemanager und Datanode eingesetzt werden.

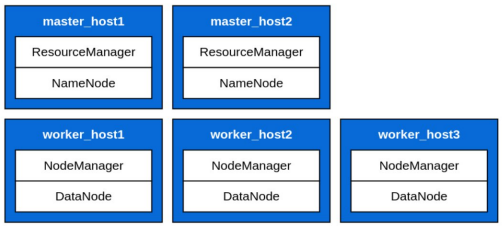

Bild 3 Hardware-Architektur

Diese Implementierung ist notwendig, um die Ressourcen des gesamten Rechners effizient zu nutzen: datanode neigt z.B. dazu, mehr Festplatten zu nutzen, während nodemanager eher das Gegenteil tut.

Damit Hadoop ohne Ausfallzeiten läuft, verwenden wir ein klassisches Schema mit Namensknoten-Synchronisierung über Quorum-Journale [1]:

- Es gibt einen aktiven Namensknoten, der Informationen schreibt;
- Es gibt Standby-Namenknoten, die Änderungsprotokolle aus den Protokollen lesen, Änderungen synchronisieren und jederzeit aktiv werden können.

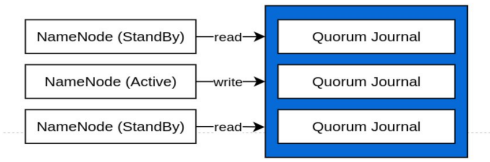

Bild 4 Synchronisationsschema

Darüber hinaus nutzt unser Hadoop die Vorteile von Verfügbarkeitszonen - jeder Block verfügt über Replikate in drei entfernten Rechenzentren. Dies stellt sicher, dass die Lösung auch bei einem vorübergehenden oder vollständigen Ausfall eines Rechenzentrums stabil bleibt.

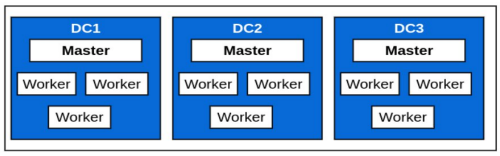

Bild 5 Einstellung der Verfügbarkeitszonen

II. Probleme bei der anfänglichen Umsetzung und Möglichkeiten zu ihrer Lösung

Unser Hadoop auf Hardware hatte einige "Schwächen", die wir beheben oder verbessern wollten.

- **Betrieb.** Die meisten Aufgaben im Zusammenhang mit der Verwaltung der Lösung und der Wartung der Infrastruktur mussten wir teilweise oder vollständig manuell durchführen. Das Ersetzen/Hinzufügen von Knoten, die Aktualisierung von Konfigurationen auf dem Cluster, die Bereitstellung neuer Cluster - viele Aspekte erforderten die Beteiligung von Spezialisten.
- **Veränderliche Komponentenzustände.** Wir verfügen über gut entwickelte Praktiken für die Arbeit mit physischer Hardware, die es uns ermöglichen, das Auftreten von variablen Zuständen zu minimieren. Gleichzeitig war es nicht möglich, Situationen vollständig zu vermeiden, in denen einige Knoten Konfigurationen oder Softwareversionen haben, die nicht mit denen anderer Knoten im Cluster übereinstimmen, was zu Zwischenfällen führen kann.
- **Kombination von Compute und Storage.** Wir wollten Compute und Storage trennen, um die Effizienz des Ressourcenverbrauchs zu verbessern. Der Stromverbrauch ist über den Tag verteilt ungleichmäßig, wobei die Rechenlast in der Nacht ihren Höhepunkt erreicht - gemäß unserem internen SLA müssen alle Berechnungen bis zum Morgen abgeschlossen sein. Infolgedessen müssen wir dem Projekt Kapazitäten mit einer großen Reserve zuweisen, die die meiste Zeit ungenutzt bleiben. Bei einem Schema mit gemeinsam genutzter Rechen- und Speicherkapazität haben wir die Möglichkeit, den Cluster in Zeiten hoher Auslastung zu erweitern und die Ressourcen in Phasen, in denen sie nicht benötigt werden, auf andere Anwendungen zu übertragen.

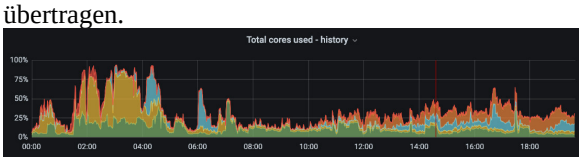

Bild 6 · Diagramm zum CPU-Verbrauch

Das heißt, wir brauchten eine Umsetzung, die es ermöglicht:

- Eine unabhängige horizontale und vertikale Skalierung von Rechen- und Speicherschichten;
- Verbesserung der Benutzerfreundlichkeit (Inbetriebnahme oder Löschung von Knoten, Erstellung neuer Cluster und andere Operationen);
- Den unveränderlichen Zustand der Komponenten zu erhalten.

Es gab drei Möglichkeiten, dies zu erreichen.

- **Übergang zu verwalteter Infrastruktur.** Dieser Ansatz beinhaltet die Nutzung von Infrastruktur und Diensten eines Drittanbieters, der die Verwaltungs- und Wartungsaufgaben übernimmt. Mit dieser Option können Sie standardmäßig Datenverarbeitung und Speicherung trennen und betriebliche Schwierigkeiten vermeiden. Für uns war sie jedoch nicht geeignet - wir können keine Nutzerdaten in öffentlichen Clouds speichern, und die Kosten für die Bereitstellung und Wartung einer solchen Lösung für unser Datenvolumen in einer Cloud eines Drittanbieters sind unverhältnismäßig hoch.
- **Wechsel des Technologie-Stacks.** Wir könnten etwas Kompatibles (Ceph, Apache Ozon, unsere eigene Lösung) für die Speicherung und Spark über K8s oder Trino über K8s für die Datenverarbeitung verwenden. Dies sind die gängigsten Lösungen. Aber in unserem Fall ist dieser Ansatz nicht gerechtfertigt: Erstens haben wir bereits hdfs, das uns gut passt, und zweitens haben wir nicht nur Spark, sondern auch andere Legacy-Tools, während Kubernetes überhaupt nicht verfügbar ist. Außerdem ist es nicht der naheliegendste Weg, den gesamten Stack zu ändern, um ein paar Vorteile zu erhalten.

- **Bereitstellung der Lösung auf der Cloud Infrastruktur.** Azure hat seinen eigenen Container-Orchestrator - Azure Kubernetes Service. Seine nächstgelegenes Open-Source-Alternative ist Kubernetes. Das heißt, wir können Hadoop in AKS bereitstellen. Und dieser Weg ist bereits gut ausgetreten - ähnliche Projekte wurden bereits von Uber [2] und Yandex umgesetzt. Für uns ist diese Option zu einer Priorität geworden - sie ist evolutionär und steht im Einklang mit der Entwicklungsstrategie des Kundenunternehmens.

III. Von der Idee bis zu den ersten Entwürfen

AKS ist ein großer Container-Orchestrator, der die Zuweisung aller Ressourcen verwaltet. Er entscheidet sogar, wo eine bestimmte Anwendungsinstanz gehostet wird. Zum Beispiel können Nodemanager und Datanode automatisch auf verschiedenen Hosts gehostet werden.

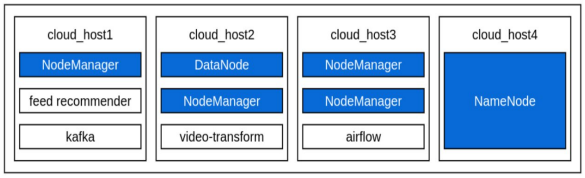

Bild 7 Optionen für die Ressourcenallokation

Die wichtigste Voraussetzung für das System ist die maximale Nutzung der verfügbaren Ressourcen. Für uns besteht der Hauptvorteil der Cloud darin, dass sie das Problem der Ressourcenverwaltung löst und es uns ermöglicht, Komponenten flexibel zu skalieren, ohne sie manuell zwischen den Hosts übertragen zu müssen.

Für die Bereitstellung eines Clusters in der Cloud genügen kleine Manifeste, die das Docker-Image, die Anzahl der Ressourcen und einige zusätzliche Parameter angeben.

```
type: service
name: nodemanager
queue: main-hadoop.hadoop.datalab.batch
availability:
  mingunning: -25
  governor: reported
alloc:
  vcores: '12'
  mem: 45G
  lan_out: 500m
  lan_in: 500m
env:
  -HADOOP_HEAPSIZE_MAX=2500m
  -HADOOP_HEAPSIZE_MIN=2500m
image:
  name: oracle9-datalab-hadoop-nm
  version: stable
  digest: sha256:02c33ac13c7e8 <....>
mounts:
  data: /mnt/hadoop/1
timeouts:
  deploy: 10m
  start: 10m
  stop: 10m
replicas: '351'
```

Code 1 Zusätzliche Parameter für Docker

```
type: storage
namespace: odkl
name: nodemanager
queue: main-
hadoop.hadoop.datalab.batch
comment: null
prealloc:
  vcores: '12'
  mem: 45G
  lan_out: 500M
  lan_in: 500M
volumes:
  data:
    size: 350G
    durability: cache
    type: hdd
```

Code 2 Zusätzliche Parameter für Docker

Auf den ersten Blick nichts Kompliziertes. Aber damit Hadoop funktioniert, mussten wir vorher noch einige Probleme lösen.

- **Konfiguration.** In der Praxis wird die Konfiguration auf unterschiedliche Art und Weise behandelt. Bei unserer Kunde wird zu diesem Zweck ein eigenes System verwendet - das Portal Management System (PMS). Das Tool ist in die gesamte Produktion integriert. Es ermöglicht Ihnen, Konfigurationen zu einer einzelnen Anwendung zu schreiben und Änderungen dieser Konfigurationen zu abonnieren. Mit PMS können Konfigurationsdateien an Hadoop-Container geliefert werden. Im Ergebnis schließen wir die Aufgaben des Konfigurationsmanagements mit dem Portal Management System.
- **Kerberos.** Hadoop verfügt über Kerberos, ein Netzwerk-Authentifizierungsprotokoll mit Vertrauensstellung. Es wird zur Authentifizierung von Cluster-Clients und Knoten innerhalb des Clusters verwendet. Tatsächlich ist dies die einzige Möglichkeit, die Sicherheit in Hadoop zu konfigurieren. In unserer Implementierung auf Bare Metal haben wir diese Prozesse halb-manuell gesteuert, aber für die Cloud haben wir sofort nach einer Möglichkeit gesucht, sie vollständig zu automatisieren. Zu diesem Zweck schrieben wir den Kerberos-Registrierungsmanager - eine Komponente, die die Topologie neuer Cluster scannt, Knoten ohne Konten findet und sie in Kerberos anlegt sowie Keytabs erstellt und sie in Vault einspeichert.

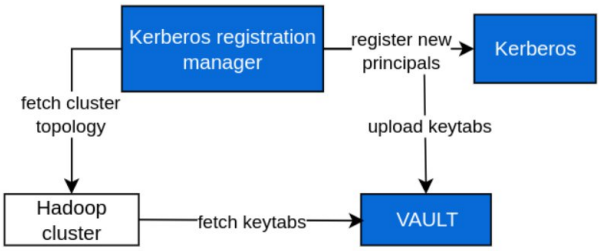

Bild 8 Übersicht über die Konfiguration des Sicherheitssystems

Zusammengefasst sieht die Pipeline zur Erstellung von Clusterkomponenten wie folgt aus:

- Durchführung einer Manifestübermittlung an die Cloud.
- Die Cloud identifiziert einen Server, der zum Starten geeignet ist. Der Container wird gestartet.
- Konfigurationen, Zertifikat und Kerberos-Keytab werden an den Container geliefert.
- Hadoop-Startup. Ein Hadoop-Knoten tritt dem Cluster bei.

So können wir einen Cluster erstellen, in dem Compute und Storage, HDFS und YARN getrennt sind.

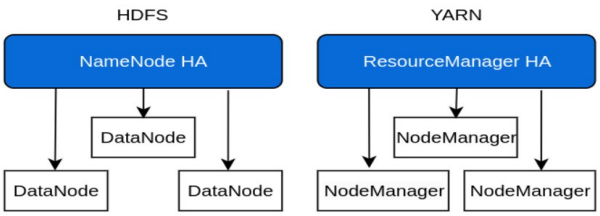

Bild 9 HDFS und YARN Einstellung

IV. Von der Theorie zu echten Petabytes

Sobald der Algorithmus für die Erstellung eines Clusters in der Cloud ausgearbeitet war, stellte sich die Frage nach der Verlagerung von Hunderten von Petabytes an Daten von physischer Hardware in die Cloud.

Es gibt zwei Hauptansätze für die Migration:

- schrittweise Migration von Anwendungen auf den neuen Cluster;
- Das schrittweise Hinzufügen von Cloud-Knoten.

V. Schrittweise Migration von Anwendungen auf den neuen Cluster

Bei dieser Methode wird ein Cloud-Cluster eingerichtet und ganze Anwendungen oder große Komponenten dorthin migriert. Es ist auch möglich, ein etwas komplexeres, aber kosteneffizienteres System zu implementieren, bei dem der Cloud-Cluster schrittweise im Verhältnis zum "Zusammenbruch" der physischen Infrastruktur erweitert wird:

- die schrittweise Übertragung des Bandes;
- Kopieren der Geschichte;

- Aufbau des Clusters, Freigabe alter Ressourcen.

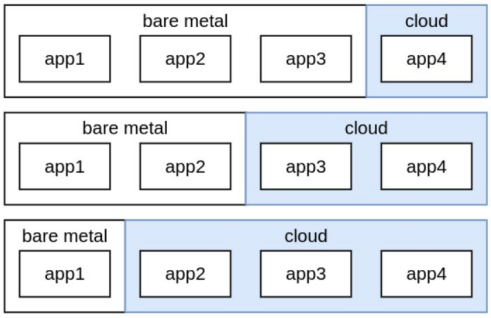

Bild 10 Migrationsplan

Die Methode ist zuverlässig, hat aber auch einige Nachteile:

- der Prozess muss manuell gesteuert und überwacht werden;
- Das Kopieren von Daten kann sehr lange dauern;
- Zum Zeitpunkt der Migration muss die Fachlogik der Datenverarbeitung neu aufgebaut werden, und die Benutzer müssen gleichzeitig mit mehreren Clustern arbeiten: Lesen in einigen Clustern, Schreiben in anderen.

Infolgedessen hat diese Option für uns nicht funktioniert.

VI. Schrittweises Hinzufügen von Cloud-Knoten

Die Methode sieht vor, dass ein neuer, leerer Datenknoten parallel zum alten Cluster in der Cloud erstellt und sofort in den Cluster aufgenommen wird.

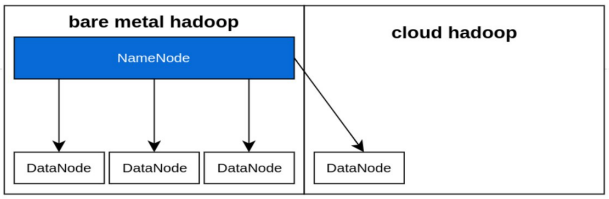

Bild 11 Schrittweises Hinzufügen von Cloud-Knoten. Schritt 1

Der Cluster beginnt, Daten auf einen neuen Knoten zu schreiben. Parallel dazu beginnt einer der alten Knoten mit der Ausgabe durch das Stilllegungsverfahren [3] (Blöcke werden nach und nach auf andere Knoten repliziert).

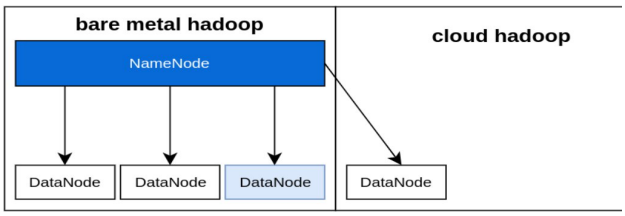

Bild 12 Schrittweises Hinzufügen von Cloud-Knoten. Schritt 2

Sobald der alte Knoten leer ist, kann er aus dem Cluster entfernt werden.

Mit diesem Algorithmus können alle Daten nahtlos in die Cloud verschoben werden, ohne den Endnutzer zu beeinträchtigen.

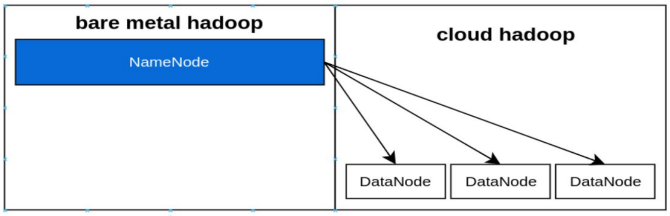

Bild 13 Schrittweises Hinzufügen von Cloud-Knoten. Schritt 3

Der letzte Schritt ist die Übertragung des Namensknotens in die Cloud.

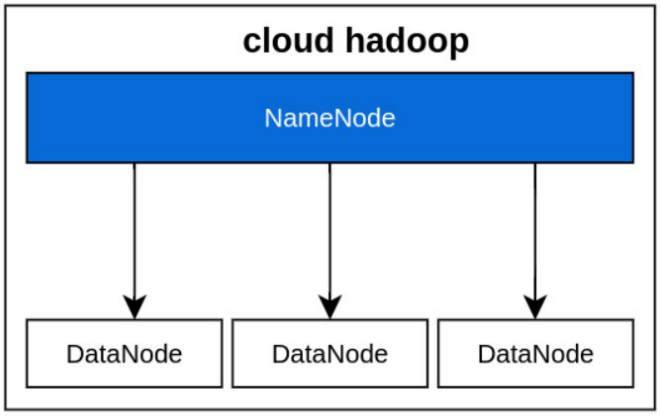

Bild 14 Schrittweises Hinzufügen von Cloud-Knoten. Letzter Schritt

Danach kann die Migration von Hadoop in die Cloud als abgeschlossen betrachtet werden. Mit diesem Ansatz werden Ausfallzeiten und Datenverluste vermieden.

Wir begannen, diesen Weg einzuschlagen, aber...

VII. Migration und erste Misserfolge bei der Erfüllung der Erwartungen

Nach dem Start der Datenmigration auf den Knoten in der Cloud haben wir eine Replikationsgeschwindigkeit von 2 Tausend Blöcken pro Minute erreicht. Das sind etwa 250 GB. In Anbetracht der Tatsache, dass wir etwa 150 Millionen Blöcke hatten, selbst in einem Cluster, der nicht der größte war, dauerte die Migration bei dieser Geschwindigkeit etwa 52 Tage, was unannehmbar lang ist.

Wir fingen an, uns damit zu beschäftigen, untersuchten Namenode und sahen, dass die Kernel nicht geladen sind, aber RedundancyMonitor im Hintergrund läuft, die Methode ist computeDatanodeWork().

```
/**
 * Periodically calls computeBlockRecoveryWork(),
 */
private class RedundancyMonitor implements Runnable {
    @Override
    public void run() {
            while (namesystem.isRunning()) {
                try {
                // Process recovery work only when active
NN is out of safe mode.
                if (isPopulatingReplQueues()) {
                    computeDatanodeWork();
                    processPendingReconstructions();
                    rescanPostponedMisreplicatedBlocks();
          lastRedundancyCycleTS.set(Time.monotonicNow());
                }

TimeUnit.MILLISECONDS.sleep(redundancyRecheckIntervalMs);
```

Code 3 Methodenaufruf - computeDatanodeWork()

Sie ist für die Zuweisung von Aufträgen zur Replikation von Knoten zuständig. Der Algorithmus zur Auswahl von Datenknoten für die Replikation sieht ungefähr wie folgt aus:

- Es werden zufällige Datenknoten in zufälligen Datenzentren ausgewählt.
- Die Einhaltung der Blockzuweisungspolitik wird überprüft - es ist wichtig, dass drei Datenknoten in drei verschiedenen Datenzentren ausgewählt werden.
- Wenn die Bedingung nicht erfüllt ist, wird die Schleife wiederholt.

Unsere Analyse zeigt, dass 90 % der Zeit für die Auswahl zufälliger Datenknoten und zufälliger Datenzentren aufgewendet wird. Dabei handelt es sich in der Regel um schnelle Vorgänge. Um die Ursache des Problems zu finden, begannen wir, die Implementierung auf Code-Ebene zu untersuchen.

Die Standardreihenfolge für die Auswahl eines Zufallsknotens ist wie folgt:

- Alle Datenknoten im Datenzentrum werden überprüft.
- Es wird geprüft, ob die richtige Art von Discs vorhanden ist.
- Wenn es Scheiben gibt, wird der Knoten in die Liste aufgenommen.
- Ein zufälliger Datenknoten wird aus der Liste ausgewählt.

```
def choice_random(block_storage_type):
    selected_datanodes = []
    for datanode in datanodes:
        if datanode.storage_type == block_storage_type:
            selected_datanodes.append(datanode)
    return get_random(selected_datanodes)
```

Code 4 Frühere Implementierung der Knotenauswahl

Bei dieser Implementierung bestand das Hauptproblem darin, zu prüfen, ob Scheiben des entsprechenden Typs vorhanden sind - wahrscheinlich wurde bei der Entwicklung nicht berücksichtigt, dass die Liste möglicherweise nicht 10, sondern z. B. 150 Einträge enthält, was den Zeitaufwand vervielfacht.

In unserer Installation gibt es jedoch nur Festplatten (HDD), so dass es nicht notwendig ist, deren Übereinstimmung zu überprüfen. Dadurch konnten wir den Code ein wenig umschreiben und das Schema erheblich vereinfachen - jetzt erfolgt die Auswahl immer nach dem Zufallsprinzip ohne zusätzliche Genehmigungen.

```
def choice_random_fast(block_storage_type):
    random_datanode = get_random(selected_datanodes)
    while (random_datanode.storage_type != block_storage_type:
        random_datanode = get_random(selected_datanodes)
    return random_datanode
```

Code 5 Neue Implementierung der Knotenauswahl

Durch diese Manipulationen konnten wir die Replikationsgeschwindigkeit auf 15 Tausend Blöcke pro Minute erhöhen (2 TB gegenüber 250 GB vor den Änderungen). Infolgedessen konnte die Migrationszeit von 52 auf 7 Tage reduziert werden.

Danach gab es noch die Frage der Master-Migration.

VIII. Migration von Mastern

Der Master-Migrationsalgorithmus ist ziemlich transparent.

- Zum Beispiel gibt es einen Cluster mit zwei Namensknoten - einer ist aktiv und der andere ist standby. Es gibt drei Protokolle, in die sie mit Quorum schreiben.

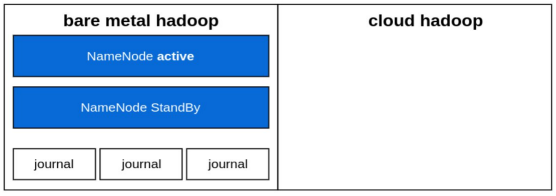

Bild 15 Master Migration Order. Schritt 1

- Ein neuer Namensknoten und ein paar Protokolle (um die Ungewöhnlichkeit zu bewahren) werden der Cloud hinzugefügt.

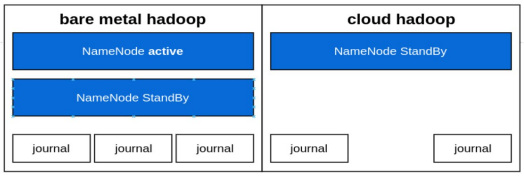

Bild 16 Master Migration Order. Schritt 2

- Nach der Überprüfung der Korrektheit wird der Namensknoten in der Cloud aktiviert.

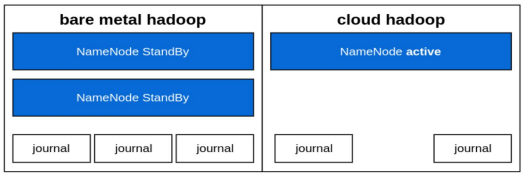

Bild 17 Master Migration Order. Schritt 3

- Einer der alten Namensknoten wird aus dem Verkehr gezogen.

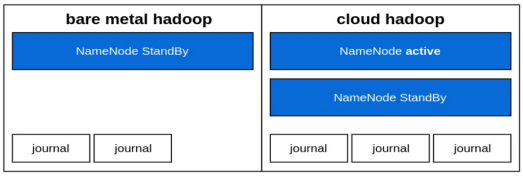

Bild 18 Master Migration Order. Schritt 4

- Der zweite Namensknoten-Standby wird nach demselben Prinzip in die Cloud übertragen. Danach wird die restliche Hardware aus dem Cluster entfernt. Die Migration des Masters ist abgeschlossen.

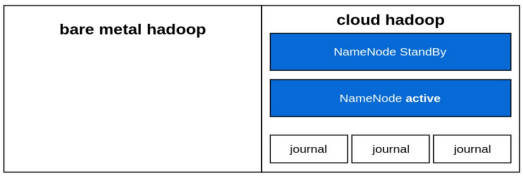

Bild 19 Master Migration Order. Letzter Schritt

Aber es gab ein paar Feinheiten, die zu beachten waren.

- Unmöglichkeit, Komponenten dynamisch hinzuzufügen. Sie können keine neuen Namensknoten und Protokolle dynamisch zu Hadoop hinzufügen. Eine solche Funktion wird für die nahe Zukunft versprochen, ist aber noch nicht verfügbar. Am Ende mussten wir die Knoten in mehreren Clustern manuell neu laden. Das ist zwar nicht kritisch, hat aber einige Unannehmlichkeiten verursacht.
- **Die Komplexität einer nahtlosen Datenmigration.** Es gibt viele Clients, die mit namenode arbeiten, und eine Unterbrechung der Kommunikation mit einem von ihnen kann zu Ausfällen auf Prozessebene führen, z. B. zur Unfähigkeit, zu lesen und zu schreiben, was zu lokalen Ausfallzeiten führt. Gleichzeitig war es uns wichtig, dass Benutzer, die mit Hadoop arbeiten, nicht von der Migration betroffen sind.

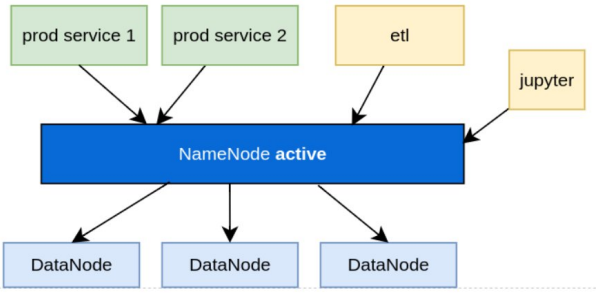

Bild 20 Client-Architektur in namenode

Die Schwierigkeit besteht darin, dass die Clustertopologie auf dem Client in der Datei hdfs-site.xml gespeichert ist.

hdfs-site.xml >

```
<name>dfs.namenode.rpc-adress.clustername.hostname1></name><value>hostname1:8020</value></property>
<name>dfs.namenode.rpc-adress.clustername.hostname2></name><value>hostname2:8020</value></property>
<name>dfs.namenode.rpc-adress.clustername.hostname3></name><value>hostname3:8020</value></property>
<name>dfs.namenode.https-adress.clustername.hostname1></name><value>hostname1:50470</value></property>
<name>dfs.namenode.https-adress.clustername.hostname2></name><value>hostname2:50470</value></property>
<name>dfs.namenode.https-adress.clustername.hostname3></name><value>hostname3:50470</value></property>
```

Bild 21 hdfs-site.xml

Wird bei der Erstellung eines neuen Cloud-Namensknotens nicht allen Clients ein neuer Cloud-Namensknoten hinzugefügt und der Client nicht entsprechend umkonfiguriert, kann das System nicht mit dem Cluster arbeiten und die Anfragen führen zu einem Standby-Namensknoten. Um solche Probleme zu vermeiden, sollten die Konfigurationen der Clustertopologie an jeden Client übermittelt und bei Änderungen aktualisiert werden.

In unserem Fall wird dieses Problem dadurch gelöst, dass die Konfigurationen in den meisten Anwendungen dynamisch angewendet und im Portal Management System gespeichert werden, das in diesem Fall als einzige Quelle der Wahrheit fungiert. Bei Hadoop haben wir alle Konfigurationen im Vorfeld zentralisiert, so dass wir nur an einer Stelle Informationen über einen neuen Knoten angeben müssen und diese automatisch in allen Cluster-Clients abgerufen werden.

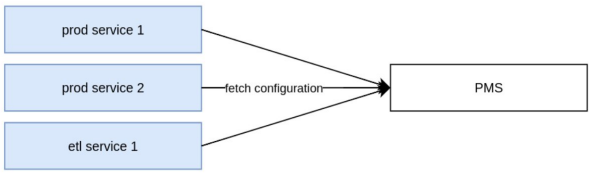

Bild 22 System zum Speichern und Aufzeichnen von Einstellungen

Danach war die Migration abgeschlossen. **Erfolg**!

IX. Leben nach der Migration

Schon bald nachdem wir mit dem Betrieb von Hadoop in der Cloud begonnen hatten, wurden wir mit zwei Umständen konfrontiert.

Erstens ist die Gesamtberechnungsleistung um 10-15 % gesunken. Der Verlust war zum Teil erwartet und vorhersehbar, da wir die lokale Datenverarbeitung aufgegeben haben und es mehr Abrufe über das Netz gibt. Für uns ist eine solche Dynamik nicht kritisch - der Leistungsverlust wird durch die Vorteile, die sich

aus der Arbeit in der Cloud ergeben, mehr als kompensiert.

Zweitens stellten wir eine allgemeine Verschlechterung der Rechenleistung fest.

Wir führen also täglich DWH-Berechnungen durch.

Bei der Arbeit mit Hadoop auf Bare Metal war es für uns die Regel, dass die Berechnungen am Vormittag abgeschlossen waren - bis maximal 11:00 Uhr. Nach der Migration in die Cloud wurde dieser Zustand beibehalten. Doch nach einiger Zeit begann eine starke Verschlechterung - das Ende der Berechnungen verschob sich zunächst auf 12:00 Uhr, dann auf 16:00 Uhr und schließlich auf 18:00 Uhr.

Und laut Statistik nehmen die Berechnungen den gesamten verfügbaren Speicherplatz in Anspruch. Und einige der Berechnungen haben nicht einmal die Zeit, von Tag zu Tag gezählt zu werden.

Bild 23 Verlauf der RAM-Nutzung

Weitere Analysen ergaben, dass die Probleme bei der IO lokalisiert sind - von einigen Datenknoten liest unser Nodemanager 10-100 mal langsamer als von anderen.

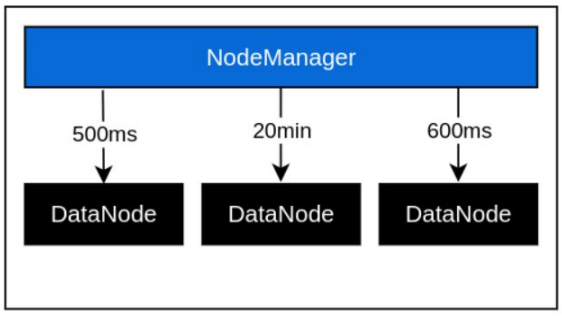

Bild 24 Ungleichmäßige Verteilung der Last auf die Knotenpunkte

Der Grund dafür war IOCost [4] - eine Technologie, die es ermöglicht, Anwendungen über eine Festplatte voneinander zu isolieren und ihnen Lese-/Schreibgarantien zu geben. So kann zum Beispiel Kafka bequem mit jeder anderen Anwendung, die die Festplatte nutzt, "benachbart" werden.

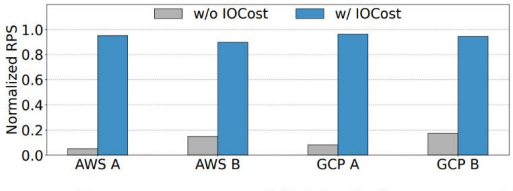

Bild 25 Abfragen pro Sekunde (RPS) eines laden-empfindlichen Workloads, wenn er mit einem Memory-Leak-Workload in AWS EBS und Google Cloud Persistent Storage gestapelt wird

In unserem Fall stellte sich heraus, dass es einen direkten Zusammenhang mit der Einbeziehung von IOCost in neue Rechenzentren gab. Die Technologie wurde schrittweise auf neue Rechenzentren ausgeweitet, um bessere Garantien zu bieten, was jedoch die Verschlechterung nur noch verschlimmerte.

Irgendwann beschlossen wir, die Änderungen rückgängig zu machen (IOCost aufzugeben), und stellten fest, dass die negativen Auswirkungen vollständig ausgeglichen wurden - die Berechnungen wurden wieder vor dem Mittagessen und sogar noch früher (um 6-7 Uhr) abgeschlossen. Hadoop wurde gemütlich.

Um auch "benachbarte Anwendungen" zu schützen, haben wir die Lösung umgestaltet und konnten IOCost für die gesamte Produktion aktivieren, ohne Hadoop und andere Anwendungen zu beeinträchtigen.

Auf diese Weise haben wir die letzte Phase der Migration abgeschlossen und sind schließlich zu Hadoop in der Cloud migriert.

X. Unsere Ergebnisse und die daraus gezogenen Lehren

Die gesamte Umstellung - von der Idee bis zur Fertigstellung - hat uns 1,5 Jahre gekostet und wurde von nur drei Spezialisten durchgeführt.

Die Umstellung auf die Cloud hat uns eine Reihe von Vorteilen gebracht. Wir:

- konnten ihre Infrastruktur flexibel skalieren und sparten N Millionen an Beschaffungskosten, da sie nur noch Discs kaufen konnten;
- die Aufgaben der Hardwareverwaltung und -wartung an das AKS-Team delegiert;
- Vollständig automatisierte Hadoop-Routineoperationen (Cluster-Erweiterung, Austausch defekter Festplatten/Maschinen und andere);
- fusionierte Computercluster, lernte nachts mehr Ressourcen zu nutzen und beschleunigte die Berechnungen;
- Wir haben Hadoop als Service entwickelt, den wir nun auch anderen Geschäftsbereichen der Holding zur Verfügung stellen.

Auf der Grundlage unserer Erfahrungen können einige Empfehlungen ausgesprochen werden:

- Scheuen Sie sich nicht, den Quellcode des Systems zu lesen und zu bearbeiten, wenn Sie dies für eine Aufgabe benötigen.
- Erfassen Sie alle Hadoop-Komponenten mit einer Vielzahl von Metriken und reagieren Sie rechtzeitig auf verdächtige Signale.
- Das Nichtverstehen von Auswirkungen ist ein potenzielles Problem, das im

ungünstigsten Moment Wirklichkeit werden kann. Es ist besser, sich sofort mit den Auswirkungen zu befassen.

XI. Anwendbarkeit auf andere Kunden

Wie man sieht, kann der Prozess der Migration einer bestehenden On-Premise-Infrastruktur gut geplant und problemlos sein und viele Vorteile bringen. Wenn Sie Fragen zu diesem Artikel haben oder mehr über die Möglichkeiten der Verwendung einer Cloud-Infrastruktur in Ihrem Fall erfahren möchten, wenden Sie sich bitte an die Autoren dieses Artikels.

Tel: +49 (0) 69 3486 7535

E-Mail: info@innovaforge.de

Web: innovaforge.de/

Quellen

[1]S. F. Apache, „HDFS High Availability“, Apache Hadoop - HDFS High Availability. [Online]. Verfügbar unter: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithNFS.html

[2]M. Mithun, S. Qifan, und Z. Shuyi, „Containerizing Apache Hadoop Infrastructure at Uber“, Engineering. Zugegriffen: 27. Januar 2024. [Online]. Verfügbar unter: https://www.uber.com/en-DE/blog/hadoop-container-blog/

[3]S. F. Apache, „HDFS DataNode Admin Guide - Decommission“. Zugegriffen: 27. Januar 2024. [Online]. Verfügbar unter: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDataNodeAdminGuide.html

[4]T. Heo u. a., „IOCost: Block Input–Output Control for Containers in Datacenters“, IEEE Micro, Bd. 43, Nr. 4, S. 80–87, 2023, doi: 10.1109/MM.2023.3277783.