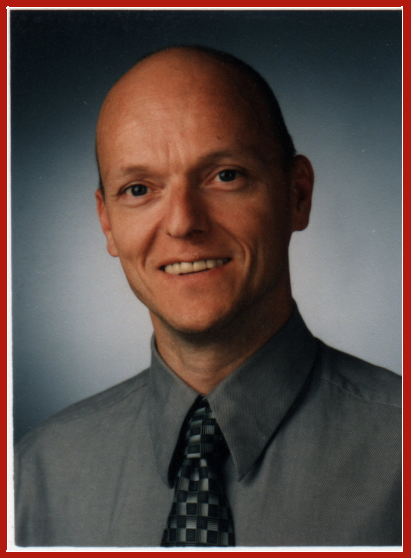
INFINISPAN / JDG

Schneller Zugriff und / oder Datenkonsistenz im Cluster

10.03.2016



Über mich

Wolf-Dieter Fink

@RedHat seit 2011

Senior Software Maintenance Engineer

Support Product Lead JBoss Data Grid

EAP Wildfly Clustering / EJB

JDG Infinispan

JAVA seit 1996

JBoss seit 2000

Java Magazin Autor

ÜBERBLICK JDG | INFINISPAN

Infinipan

- Community Project
  - http://infinispan.org
- Kein professioneller Support
- Schnellere Release
- Nutze JGroups für Kommuniktion

JDG

- Basiert auf Infinispan
- Professioneller Support @redhat.com
- Langsamere Release Zyklen
- Längerer Maintenance
  - https://access.redhat.com/support/policy/updates/jboss_notes/

ÜBERBLICK JDG | INFINISPAN

- Key Value Store (NoSQL)
  - 'HashMap' mit weiteren Features
- Distributed oder Replicated im Cluster
- Automatische Eviction und Expiration
- Querying; continues Queries
- Listeners / Events
- Near Caching für HotRod clients
- Distributed Invocation
- Kommende Änderungen
  - Map/Reduce (JDG 6.x und Infinispan 7)
    - ersetzt durch 'Distributed' Java8 Streams (ISPN 8+)
  - Server seitige scripts

ÜBERBLICK JDG | INFINISPAN

- Embedded (Library) Mode
  - Cache in der gleichen JVM wie die Anwendung
  - Schneller in-Memory Zugriff
  - Unterstützung von 2PC Transaktionen
  - Configuration in der Anwendung mittels XML oder Programatisch
  - JCache interface verfügbar

ÜBERBLICK JDG | INFINISPAN

- Client / Server Mode
  - Komplett getrennt von der Anwendung
  - Eigene Konfiguration über XML oder Management Interface
- Hot Rod Client
  - Intelligenter Zugriff über ConsistentHash
  - Java C++ C# (next node.js)
- Rest
- Memcache

PROBLEMATIK IM BETRIEB

- Änderung der Node Anzahl
  - Start (Join)
  - Stop (Leave)
- State Transfer
  - updates im Cluster
- Garbage Collection
- Network

Verhalten bei geplanter Änderung

Node

- Informiert den Cluster-Coordinator
- Speichert oder läd Daten wenn PersistentStore vorhanden

Cluster Coordinator

- Startet rebalancing für Distributed Caches
  - Key's werden anhand des ConsistentHash und numOwner neu (Segmentweise) verteilt
- Protokolliert Start und Ende des Rebalancing
- Setzt neue "Stabile" Cluster-View

Probleme bei geplanten (re)starts

- Verzögerte Response beim Zugriff
  - Daten werden ggf. von anderen Nodes angefordert
- Zu viele Daten im Cache
  - Timeout für initialen StateTransfer
- Massive Full GC's
- Netzwerk Latenz oder Überlastung

Problematiken zur Laufzeit

- Garbage Collection dauert länger als eingestellte JGroups failure detection
  - Split-Brain -> Rebalancing
- Nicht genügend Threads um die Kommunikation sicherzustellen
  - Timeouts oder lange Laufzeiten beim Sync/Async
  - Im Sync-Mode schlägt der Update fehl
  - Im Async-Mode entstehen Inkonsistenzen da nicht wiederholt wird
- Netzwerk Ausfall
  - Split-Brain - Rebalancing

Abhilfen und Lösungen
1. Garbage Collection

- Aktuelle Java VM verwenden
- JVM default Heap passt nicht!
  - -Xms == -Xmx (kein adaptives sizing)
  - Heap vergrößern da Cache-Daten länger leben
  - Old Genetation für Daten im Cache
    Empfehlung max 50-60% Nutzung
- Immer GC logging einschalten und prüfen
- Large pages -XX:+UseLargePages

Abhilfen und Lösungen
1. Garbage Collection (CMS)

- -XX:NewSize -> 500M...2GB
  - (NewGen klein halten abhängig von der Anwendung)
- CMS Balance problem
  - Klein -> möglicherweise weniger Durchsatz
  - Groß -> Probleme durch verschieben temporärer Daten -> Old
  - Vergrößern des Survivor space -XX:SurvivorRatio=16 (32)
  - CMS Probleme (parallel CMS failures)
    - früher starten (parallel)

      -XX:CMSInitiatingOccupancyFraction=60 -XX:+UseCMSInitiatingOccupancyOnly
- Aktivieren PermGen collection

  -XX:+CMSPermGenSweepingEnabled -XX:+CMSClassUnloadingEnabled

Abhilfen und Lösungen
1. Garbage Collection (G1)

- Festlegen der Pausenzeiten (als Startwert) –XX:MaxGCPauseMillis
  - 500ms für 32GB; 1000ms für 64GB
  - Größer >> Besserer Durchsatz
- Probleme mit FullGC oder "to space overflow/exhausted"
  - Größere MaxGCPauseMillis wählen
  - HEAP vergrößern
  - Ändern von -XX:InitiatingHeapOccupancyPercent (default 45)
    - Nicht kleiner als der Prozentsatz von Daten!
  - Vergrößern von -XXConcGCThreads
- Aktivieren PermGen collection
- -XX:+CMSPermGenSweepingEnabled -XX:+CMSClassUnloadingEnabled

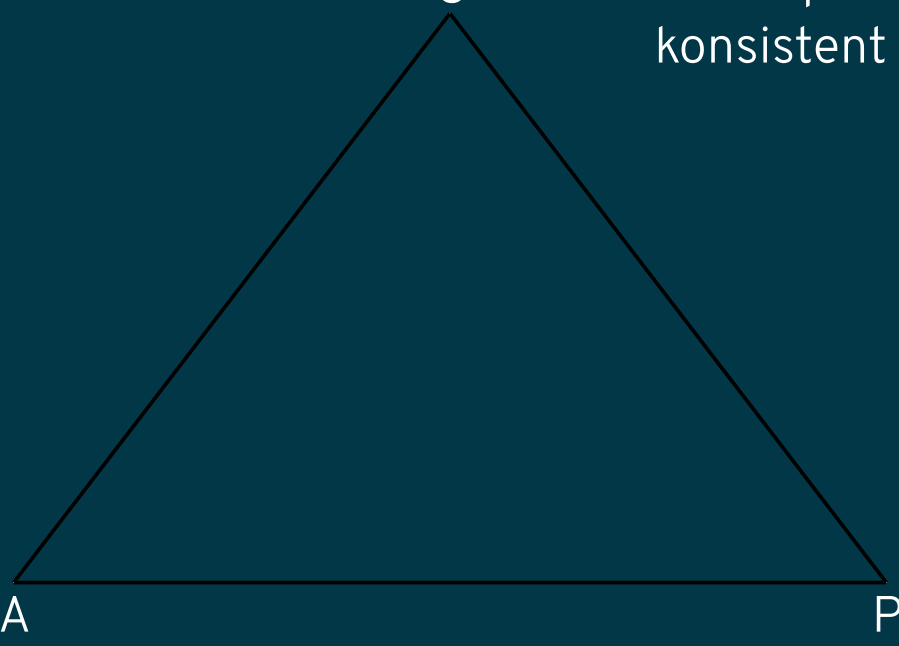
Abhilfen und Lösungen
2. Netzwerk

- UDP Multicast
  - JGroups tests (in jgroups.jar enthalten)
    - MCastSender, mehrere MCastReceiver
    - ClusterTest
- IPv4 IPv6 missmatch
  - explizit -Djava.net.preferIPv4Stack=true||false
  - Bind Adresse setzen (nicht 0.0.0.0)
- JGroups TCP Stack ist nicht für große Cluster geeignet (<10) !!

Abhilfen und Lösungen
3. Konfiguration

- REPL ist nicht für große Cluster geeignet
  - Empfehlung max. 10 Nodes
- JGroups OOB Threads werden genutzt für StateTransfer
  Bis JDG 6.3 oder Infinispan7
  - JGroups ClusterMessages gehen verloren (Heartbeat, Alive)
  - Überlastung führt zu message drop

CAP Theorem (wikipedia)



Alle Replikate werden konsistent verändert

Availablility

Anfragen werden stets beantwortet

Partition tolerance

Bei Teilausfall oder Partitionierung arbeitet das System weiter

Split Brain

Verhalten bis JDG6.3 Infinispan 7.0

- Split
  - Rebalance für jede Partition
  - Einträge können in jeder beliebigen Partition vorgenommen werden
    - Unterschiedliche Updates in verschiedenen Partitions

- Merge
  - Neue PrimaryOwner werden bestimmt, ohne Sync
  - Einträge auf Nodes die nicht Owner sind werden gelöscht!
  - Inkonsistenz möglich
    - 1 ... <numOwner> Varianten

Split Brain

Verhalten ab JDG6.4 Infinispan 7.1

- Split
  - Nicht mehr verfügbare Nodes werden aus dem CH gelöscht
    - Kein Unterschied ob Shutdown oder Crash
  - Zwei oder mehr Partitionen können die Einträge unterschiedlich Ändern
  - Änderungen die vorgenommen wurden bevor der Split erkannt wurde werden eventuell nur Teilweise ausgeführt

- Merge
  - Partition mit der höchsten ID gewinnt, alle anderen werden gelöscht!
  - Nach erzeugen eines neuen CH werden die Einträge wieder anhand von numOwners verteilt
  - Datenverlust!
  - Keine Möglichkeit individuell Einträge zusammenzuführen
  - Unter Umständen hoher Traffic

Partition Handling

(ab JDG 6.4 Infinispan 7.1)

- Wird pro Cache konfiguriert
- Abhängig von der Anzahl der Nodes wird eine Partition AVAILABLE oder DEGRADED im Falle eines Split

  - Eine Partition wird AVAILABLE wenn ...
    - Die Majorität der letzen "Stabilen View" enthalten ist
    - Weniger als numOwners die View verlassen haben
    - Es kann maximal nur eine geben!!

  - Eine Partition wird DEGRADED wenn ...
    - Alle Owner eines Segments die View verlassen
    - Nicht die Majorität der Nodes enthalten ist

Partition Handling

(ab JDG 6.4 Infinispan 7.1)

- Die Zeitspanne zwischen physikalischem und logischem Split
  - hängt ab von JGroups timeouts (FD* protokoll)
- Ein get() während dieser Zeit kann alte Werte lesen
- Ein put() kann je nach Einstellung (async) teilweise ausgeführt werden
- Transaktionen in der Zeitspanne können auf manchen Nodes committed worden sein

Partition Handling

(ab JDG 6.4 Infinispan 7.1)

Eine Partition im Mode AVAILABLE

- Erzeugt einen neuen CH und startet einen StateTransfer
- Nach erfolgtem StateTransfer ist die Partition die neue "Stabile View"
- Der Cache arbeitet wie gewohnt weiter

Partition Handling

(ab JDG 6.4 Infinispan 7.1)

Eine Partition im Mode DEGRADED

- Erzeugt KEINEN neuen CH
- Startet KEINEN StateTransfer
- get/put sind möglich wenn alle Owner eines Keys in dieser Partition liegen
  - Das gilt auch für das Anlegen oder Löschen von Einträgen
- Befinden sich nicht alle Owner in der Partition wird eine AvailablilityException geworfen
  (extends RuntimeException)

Partition Handling

(ab JDG 6.4 Infinispan 7.1)

Merge

AVAILABLE <--> DEGRAGED

- DEGRADED wird gelöscht
- Koordinator started re-hash
- Wird neue "Stabile View"

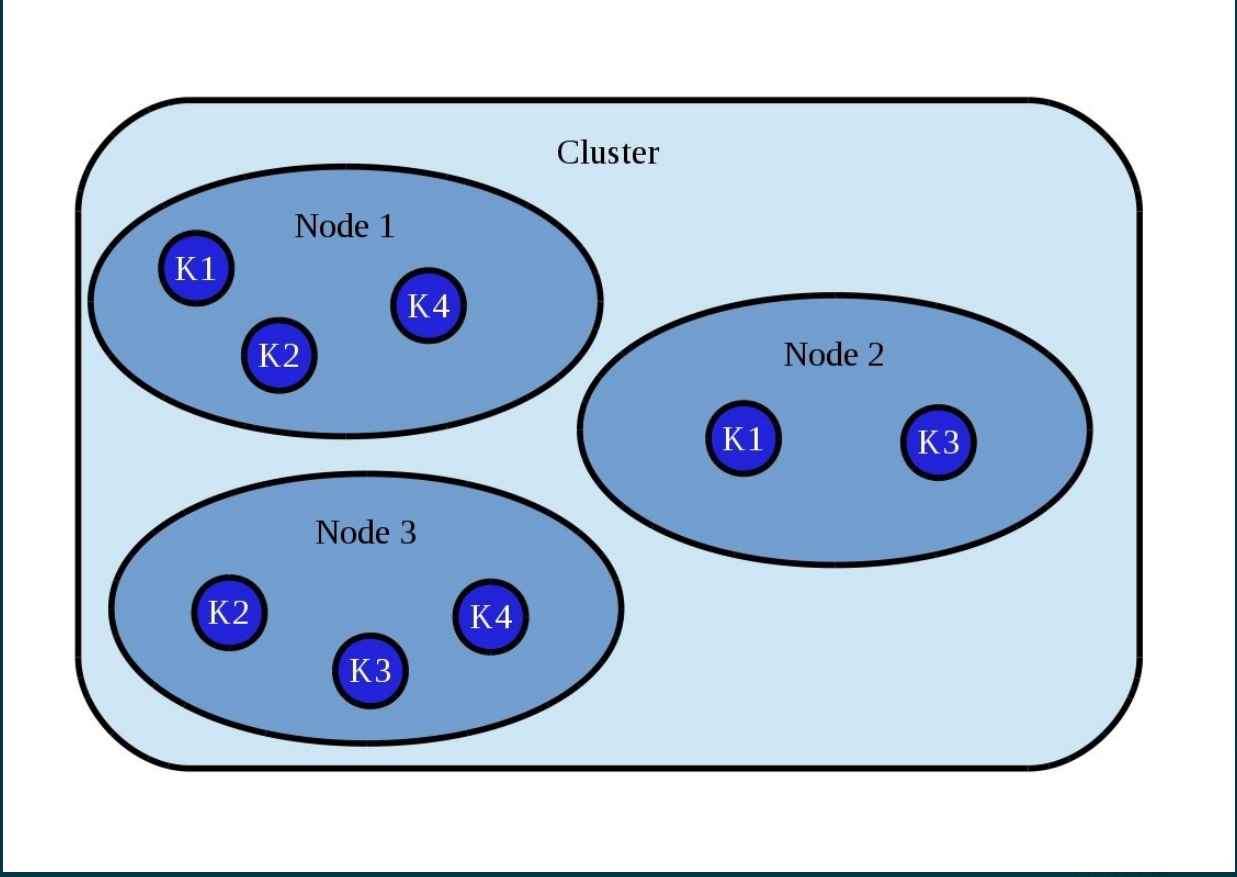
DEGRADED <--> DEGRADED

- Summe beider Partitionen erfüllt nicht die Bedingung für AVAILABLE
  - Nichts, immer noch DEGRADED
- Summe beider Partitionen erfüllt die Bedingung für AVAILABLE
  - Koordinator started re-hash
  - Wird AVAILABLE und neue "Stabile View"

Partition Handling

Beispiel 3 Nodes 2 Owner

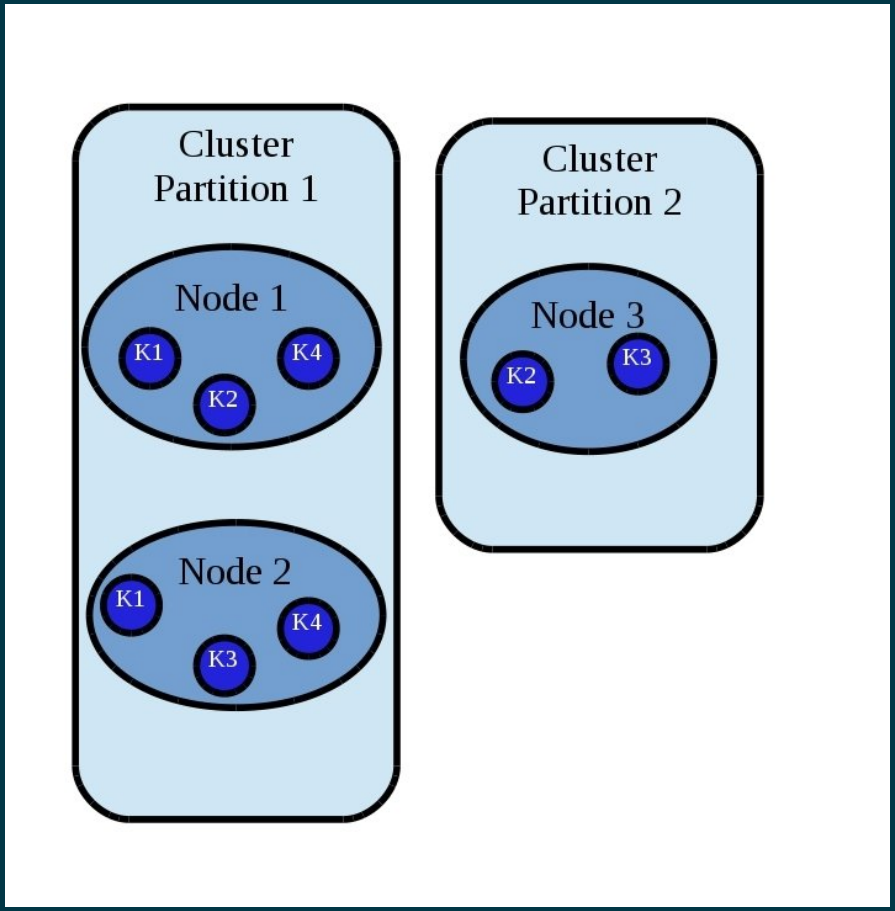
Cluster im Normalzustand



Partition Handling

Beispiel 3 Nodes 2 Owner

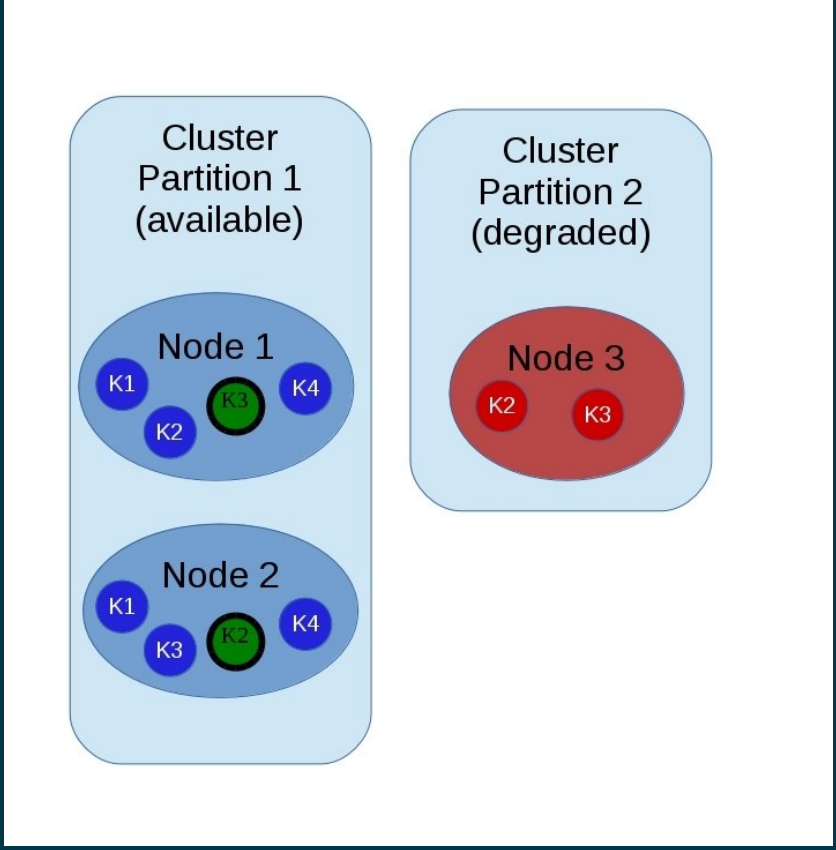
Cluster nach physikalischem Split



Partition Handling

Beispiel 3 Nodes 2 Owner

Cluster nach logischem Split



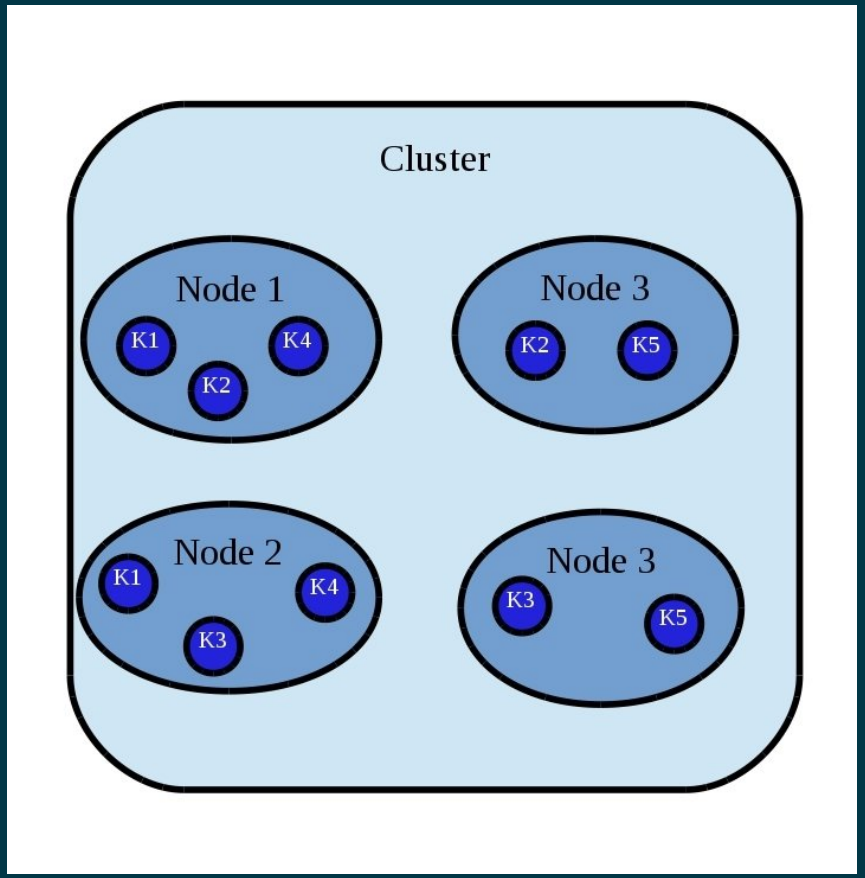
- Partition 1
  - full access
- Partition 2
  - no access
  - always AvailabilityException
  - join clear the partition

- Embedded
  - Nur Anwendungen in N1/N2 funktionieren
- Client/Server (remote access)
  - Funktional
  - Seltener Fall das der View update nur N3 enthält

Partition Handling

Beispiel 4 Nodes 2 Owner

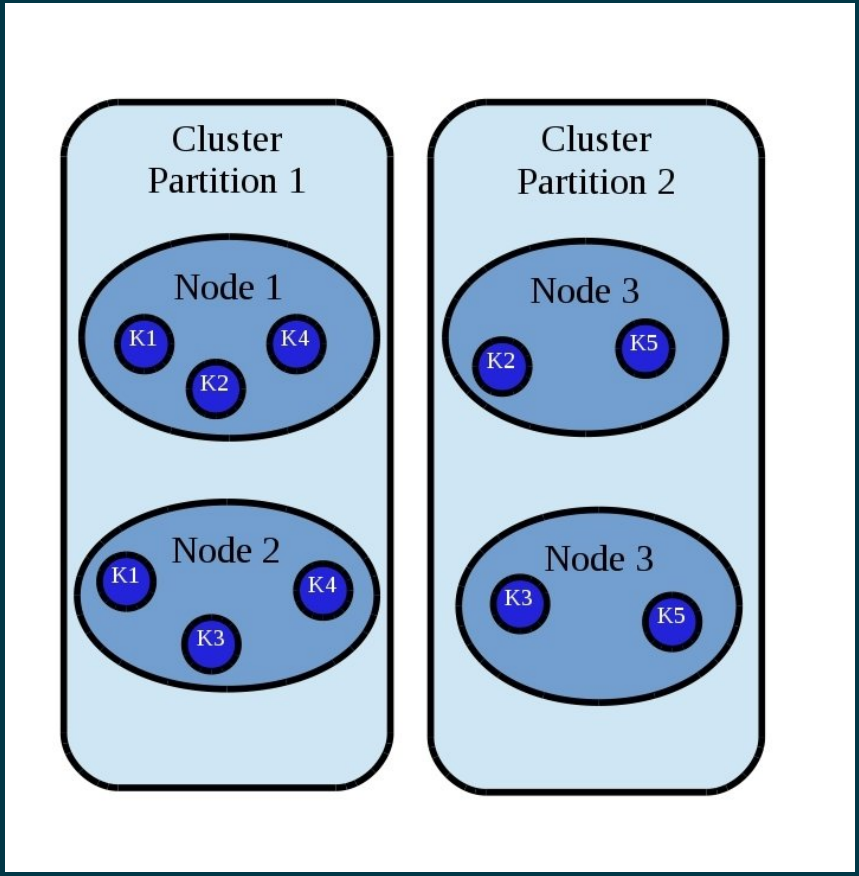
Cluster im Normalzustand



Partition Handling

Beispiel 4 Nodes 2 Owner

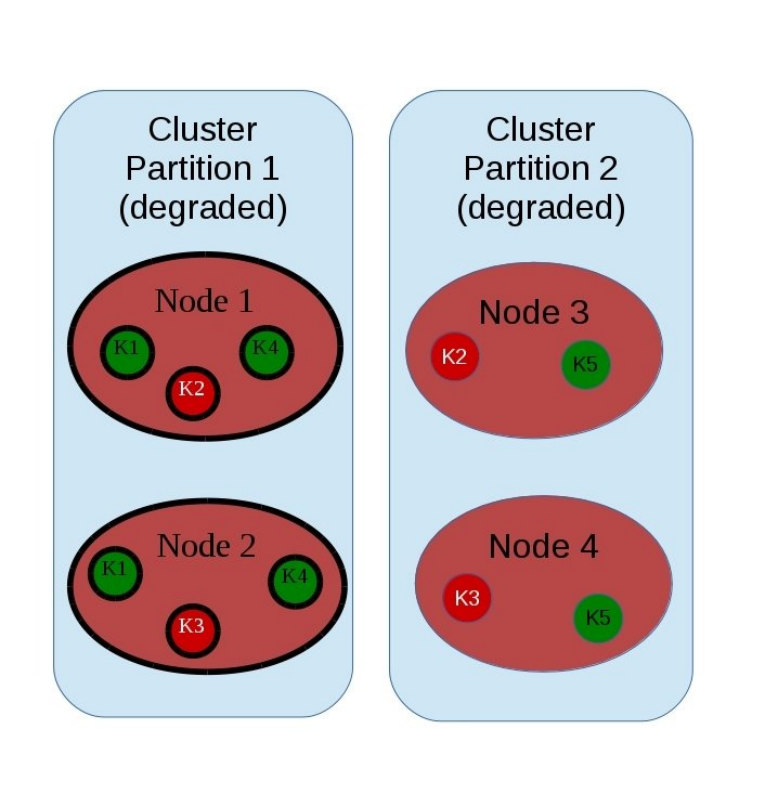
Cluster nach physikalischem Split



Partition Handling

Beispiel 4 Nodes 2 Owner

Cluster nach logischem Split



- Beide Partitionen
  - Zugriff auf 'grün'
  - AvailabilityException 'rot'
- Embedded
  - nur lokaler Zugriff
    - entweder K1 K4
    - oder K5
- Client/Server (remote access)
  - Oft beide Partitions enthalten
  - Zugriff auf K1 K4 K5

Partition Handling

Was ist mit einem replizierenden Cache ?

- Was passiert wenn ein REPL cache in 2/2 nodes getrennt wird?
- Alle sind DEGRADED es ist kein key mehr erreichbar!
- Was ist wenn ein REPL cache in 1/3 nodes getrennt wird?
- Die einzelne Node is DEGRADED und kein Key erreichbar!
- Die anderen Nodes führen einen ReHash durch und sind komplett erreichbar!

Partition Handling - Fazit

- Partition Handling ist keine Ideale Lösung für alle Anwendungen
- GC und Environment Tuning ist unerläßlich
- Anwendung muss ggf. für PH angepasst werden um auf AvailabilityExceptions zu reagieren
- Nachteile im embedded Mode da weniger Einträge zu erreichen sind als im Client/Server Modus
- PH verfügbar ab JDG 6.4 | Infinispan 7.1
- Weitere Verbesserungen ab JDG 6.5 | Infinispan 8

Partition Handling - Desaster

- Partieller Totalausfall von Nodes nach einem Split
  - Werden Nodes neu gestarted in dem Status werden diese als neu erkannt und bleiben DEGRADED
  - Die nicht mehr vorhandene Instanz bleibt im CH eingetragen
  - In diese Situation muss ein administrativer Eingriff manuell erfolgen

Partition Handling - Hilfreiches

- org.infinispan.partitionhandling.impl.PartitionHandlingManagerImpl
  - ERROR -> abgelehnten Zugriff (nur target)
  - DEBUG -> Mode Änderungen per Cache
  - TRACE -> Zugriff und Status per Key

- Manuelle Änderung des Cache Mode
  - JMX

    jboss.infinispan.Cache.<name>."clustered".Cache.Attributes.cacheAvailability
  - CLI

    /subsystem=infinispan/cache-container=clustered/distributed-cache=default:write-attribute(name=cache-availability, value=AVAILABLE)
  - Achtung! Inkonsistente Änderungen
    (z.B. zwei AVAILABLE partitions)
    Führen zu unerwartetem Verhalten beim Merge

Partition Handling

Dokumentation

- Java Magazin 9/2015
- JDG Administration Guide (6.5)
  31. Handling Network Partitions

Partition Handling - Ausblick

Zükünftige Verbesserungen

- verbesserter Merge ISPN-5290
- Wiedererkennung von Nodes nach Restart ISPN-6187
- Verschiedene Modi für Partition Handling
  - Read-Only ISPN-5511
- Callback API für Merge customizing (PRODMGT-1453)

redhat.

Noch Fragen?

Vielen Dank !