2ndQuadrant®
PostgreSQL

Harald Armin Massa

Hochverfügbarkeit mit PostgreSQL - ein Überblick

Swiss PG Day
2017-06-30
Rapperswil

Begriffe

Hochverfügbarkeit, die

Fähigkeit eines Systems,
trotz Ausfall einzelner seiner Komponenten
einen fortgeführten Betrieb zu gewährleisten.

Klassifizierungen

Verfügbarkeiten nach Harvard Research Group (Availability Environment Classification (AEC)

Conventional (AEC-0): Funktion kann unterbrochen werden, Datenintegrität ist nicht essenziell.

Highly Reliable (AEC-1): Funktion kann unterbrochen werden, Datenintegrität muss jedoch gewährleistet sein.

High Availability (AEC-2): Funktion darf nur innerhalb festgelegter Zeiten minimal unterbrochen werden.

Fault Resilient (AEC-3): Funktion muss innerhalb festgelegter Zeiten ununterbrochen aufrechterhalten werden.

Fault Tolerant (AEC-4): Funktion muss ununterbrochen aufrechterhalten werden (24/7)

Disaster Tolerant (AEC-5): Funktion muss unter allen Umständen verfügbar sein.

Klassifizierungen

Verfügbar-keits-klasse	Bezeichnung	Verfügbarkeit in Prozent	Downtime pro Jahr
2	stabil	99	3,7 Tage
3	verfügbar	99,9	8,8 Stunden
4	hochverfügbar	99,99	52,2 Minuten
5	fehlerunempfindlich	99,999	5,3 Minuten
6	fehlertolerant	99,9999	32 Sekunden
7	fehlerresistent	99,99999	3 Sekunden

Begriffe

- Recovery Time Objective RTO
  - Wie lange darf ein System ausfallen?
    Zeit zwischen Ausfall und Wiederherstellung der Betriebs

- Recovery Point Objective RPO
  - Wieviel Datenverlust kann in Kauf genommen werden?

Berechnungen

- Arbeitskosten Schweiz (2014): 59,6 Franken / Stunde (ca. 55 EUR)
- 100 Mitarbeiter, 8 Stunden
  - Ein Tag Arbeitsausfall

    8 * 100 * 59,6 = 47680 Franken Schaden
- Aufwendungen / Schaden durch
  - Umsatzausfall
  - Reputationsverlust
  - Schadenersatz

Vorgehensweisen

- Geplante Unterbrechnungen minimieren
  - Versionsupgrades!
- Ungeplante Unterbrechungen minimieren
- Redundanz für kritische Systemkomponenten

Begriffe

- Master
  - Server mit Read + Write Zugriff
  - Führendes System
- Slave (Replika / Replikant)
  - Folgt dem Master
  - Folgendes System
- Switchover
  - “freiwillige” Umschaltung, folgendes System wird zum führenden System
- Failover
  - “unfreiwillige” Umschaltung aufgrund von Fehlersituation

Redundanz kritischer Systemkomponenten

Systemkomponenten:

- Netzwerk
- Stromversorgung
- Prozessor + Hauptspeicher
- "Festplatte"

WARNUNG!
https://www.postgresql.org/docs/9.6/static/different-replication-solutions.html#HIGH-AVAILABILITY-MATRIX

2ndQuadrant PostgreSQL

Redundanz 1 – nur Prozessor und Hauptspeicher mehrfach vorhanden

- shared disk
- Bei PostgreSQL als shared disk failover möglich

  \+ nur ein Festspeicher erforderlich

  \- es kann nur einen geben:
  Sicherstellen, dass nur ein Server auf Daten zugreift

  \- Upgrades unterbrechen Systembetrieb

  Wunsch oft aus Oracle-Welt

-

PostgreSQL

2ndQuadrant PostgreSQL

Redundanz 2
Duplikation der Festplatten Blockdevice oder Hardware

- DRBD (Distributed Replicated Block Device)
- sowie proprietäre Hardware

+ mehrere Prozessoren + Hauptspeicher

+ tatsächliche mehrere Plattenspeicher verfügbar

+ non-Database files können Hochverfügbar werden (Bilder, Videos..)

2ndQuadrant PostgreSQL

Redundanz 2
Duplikation der Festplatten Blockdevice oder Hardware

- Server an duplizierter Platte warten nur
- Upgrades unterbrechen Systemverfügbarkeit
- Duplikation weiß nichts über Datenbank Schreibvorgänge

Schreibvorgänge der Datenbank = kritischer Pfad

multiple Blöcke mit wechselseitigen Abhängigkeiten:

heap, index files, visibility map,
clog, pg_subtrans, pg_multixact

•

PostgreSQL

2ndQuadrant PostgreSQL

Redundanz 3
Statement Basierte Replikation

- Üblich in MySQL Welt
- Umsetzung bei PostgreSQL mit middleware pg_pool

+ keine Belastung des Masters durch Replikanten

+ Master - Master möglich, alle Server als Schreibzugriff verfügbar

~ Unterstützung von Versionsupgrade fragwürdig / nicht vorhanden

- Daten der Server sind nicht Deckungsgleich:

a) insert into logtable (when, what) (now(), 'something dumb')

b) timing bei relationen Abhängigkeiten

Redundanz 4
Trigger basierte Replikation

- Slony, Londiste

  Änderungen werden per Trigger in Queue-Tabelle geschrieben

  Replikation der Änderungen an Slave

2ndQuadrant PostgreSQL

Redundanz 4
Trigger basierte Replikation

+ multiple Festplatten
+ multiple Server
+ Slaves können für Reads genutzt werden
+ Granularität auf Tabellenebene möglich
+ deckungsgleiche Daten
+ vacuum / analyze / reindex belasten nicht datentransfer
+ reduktion downtime minor + major upgrades

~ Tabellen müssen Primary Key haben

2ndQuadrant®
PostgreSQL

Redundanz 4
Trigger basierte Replikation

- erhebliche Belastung Master
- multiple Schreiblast (WAL, Tabelle, Queue-Table, WAL von Queue-Table)
- Trigger auf Tabellen erhöhen DB-Komplexität
- DDL herausfordernd in Replikation

Redundanz 5
Binary Replication

- logshipping
- streaming
- synchronous streaming

Redundanz 5
Binary Replication

+ multiple Festplatten
+ multiple Server
+ slaves können für Reads genutzt werden
+ geringe Belastung Master
+ reduktion downtime minor updates
+ Tabellen ohne Primary Key können repliziert werden
+ keine Herausforderungen für DDL
+ bewährt seit PostgreSQL 9.0

2ndQuadrant PostgreSQL

Redundanz 5
Binary Replication

- Granularität = alle DB des Servers
- VACUUM, ANALYZE, REINDEX
  => Belastung Datentransfer

PostgreSQL

Redundanz 6
logical replication

- pg_logical (released)
- BDR (1.0 released for 9.4,
  2.0 for 9.6 private beta)

2ndQuadrant PostgreSQL

Redundanz 6
logical replication

+ multiple Festplatten

+ multiple Server

+ Slaves können für Reads genutzt werden

+ geringe Belastung Master (minimal mehr als binary Replikation)

+ Reduktion downtime minor updates **+ major upgrades**

+ Tabellen-Granularität

+ VACUUM, ANALYZE, REINDEX = geringer Datentransfer

~ Primary Key muss vorhanden seind
- DDL herausfordernd

- neuer Code (ab 9.4)

Orthogonale Fragen

- Wie einrichten ?
- Wie umschalten ?
- Wann umschalten ?

Tools – Focus auf Binary Replication

Einrichten + Umschalten geht mit Bordmitteln:

pg_basebackup zum Einrichten Replikant

Umschalten:

- pg_ctl promote
- oder Triggerfile für den Switchover

Umsetzung

Umsetzung mit Skripten möglich, Pseudocode:

```
while test_if_server_is_alive():
      noop

promote slave to new master
reroute access
```

- Herausforderung:
  - bei jeder Umsetzung neue Verhalten
  - Wie test_if_server_is_alive() umsetzen?

Toolkits

- Pacemaker
- patroni + etcd / consul / zookeeper
- repmgr

Pro + Con tools

Pacemaker, Patroni:

+ basierend auf distributed storage

+ Nutzen von fortgeschrittenen Konsens-Algorythmen

- diverse zusätzliche pakete

Repmgr

+ standalone tool, minimale Abhängigkeiten

+ schlichter Konsensalgorhythmus

- distributed storage der Konfigs per Filecopy der repmgr.ini

- schlichter Konsensalgorhythmus

Ende.