Ansätze zur Performanzoptimierung von REST-basierten Diensten

ARVIDA Statustagung Kiel 26.10.2016

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung
DLR Projektträger

Christian Vogelgesang, DFKI

ARVIDA

Lessons Learned

- Verständnis von Daten und Zugriffsmuster essentiell wichtig

- Profiling!
  - Hotspots fast immer unerwartet
  - REST Calls sind Funktionsaufrufe!
  - Profiling von REST Anwendungen immer noch schwierig

- Einzige Problem-Konstante: Virenscanner

Generelle Optimierung

Optimierungen von konkretem Anwendungsfall abhängig

- Algorithmen auf großen Datensätzen fast immer parallelisierbar
  - Typische Beispiele
    - Exporter/Importer
    - Konverter
    - Datenaggregation
- Parallelisierung durch Auslagerung in einen oder mehreren Micro-Services
  - Ausführung unabhängig von Clients
  - Vorausschauende Ausführung möglich
- Latenzoptimierung für Spezialfälle
  - Protokoll-Ebene
  - Daten

ARVIDA

Micro-Services

Kapseln eine spezifische Funktionalität in einem „kleinen“ Service

Vorteile:

- Definierte Aufgabe
- Hohe Wiederverwendbarkeit
- Hohe Wartbarkeit
- Hohe Testbarkeit
- Vereinfachte Optimierung
- Skalierbar

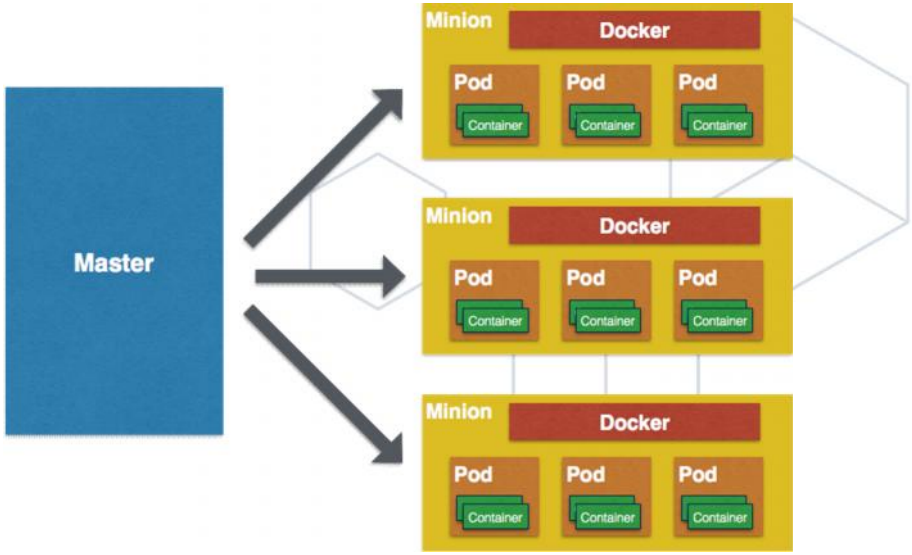
Nachteil:

- Erhöhung der minimalen Latenz

Parallelisierung von Microservices

Moderne Virtualisierung erlaubt einfaches Deployment

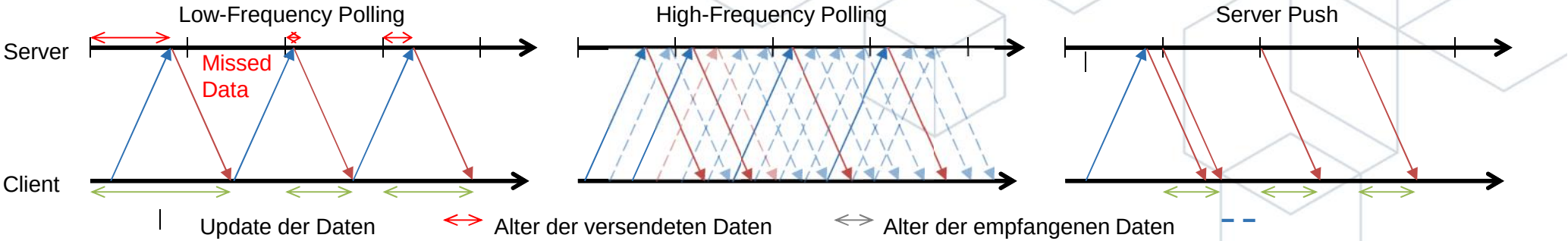
- Isolation in leichtgewichtigen Containern (z.B. Docker)
  - Anwendung mit passender Betriebssystemsinstallation
- Clustermanager (z.B. Kubernetes) erlauben Verwaltung von Containern im großen Maßstab



Latenzoptimierung im ARVIDA Kontext

- Übertragung von Trackingposen das prominenteste Beispiel
- Hohe Anforderungen:
  - Gesamt-Latenz <10ms
  - Hohe Frequenz: 60+Hz

- Serverpush wichtigste Optimierung
  - Neue Trackingdaten werden direkt an Client gesendet



Optimierung auf Protokoll-Ebene

- REST Services basieren auf HTTP Kommunikation

- Serverpush zur Latenzminimierung
  - Webhooks:
    - Server sendet aktuelle Daten automatisch zu REST Interface des Clients
  - Websockets
    - Bidirektionale TCP Verbindung

- Neue Protokolle während der Projektlaufzeit:
  - HTTP/2
  - Google QUIC

ARVIDA

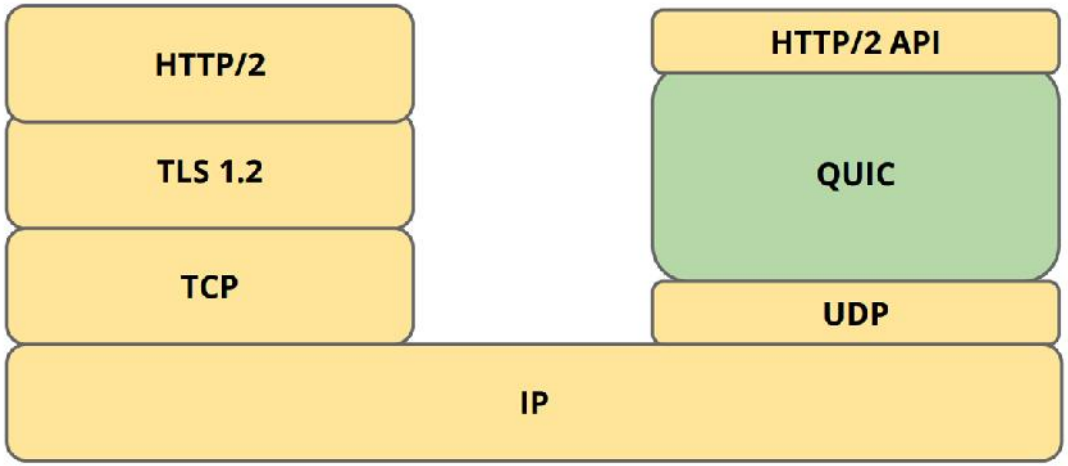
HTTP/2

Neue Revision des HTTP 2 Protokolls

- Finale Spezifikation im Mai 2015 veröffentlicht
- Wichtigste Features:
  - Komprimierter HTTP Header
  - Pipelining und Multiplexing von Requests über eine einzelne Verbindung
  - Bidirektionale Verbindungen (Server Push)
  - Abwärtskompatibel
- Durch fehlende Bibliotheken zu spät für Projekt
  - Aktuell zwei C/C++ Crossplattform (Win/Linux) Bibliotheken
    - Nghttp2 (https://nghttp2.org/)
    - H2O (https://github.com/h2o/h2o)
      - Windows Port von DFKI: https://github.com/dfki-asr/h2o-port-MSVC

QUIC

- Neues „experimentelles“ Transportprotokoll von Google
- Nutzung von UDP statt TCP
- Multiplexing auf Transportlayer
- Verbesserte Latenz
- Mai 2015:
  50% des Traffics von Chrome
  zu Google-Diensten über QUIC*
- IETF Standardisierung angestrebt



* https://blog.chromium.org/2015/04/a-quic-update-on-googles-experimental.html

FastRDF

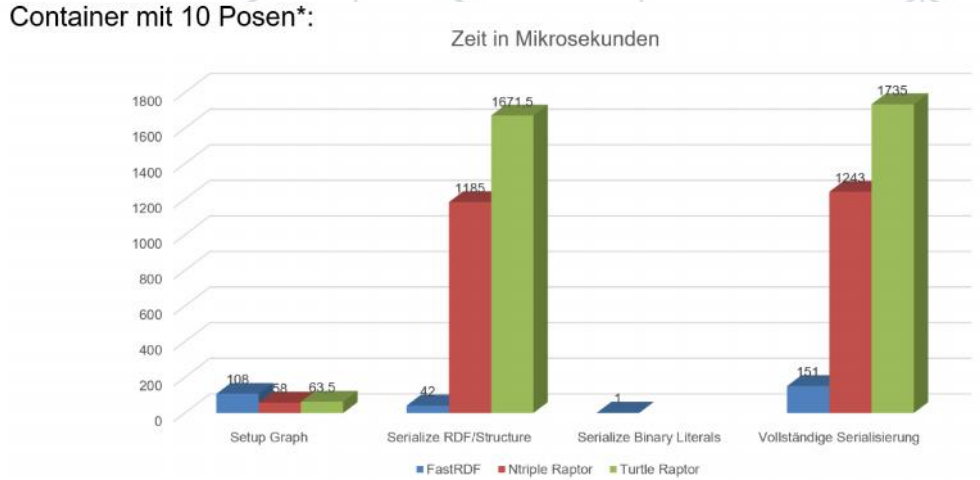
- Basiert auf den Erkenntnissen von Performance-Analysen in ARVIDA
- Trennt Darstellung der semantische Struktur von RDF Graphen von den Nutzdaten
  - Erlaubt unterschiedliche Update-Frequenzen für Graph Struktur und Literalwerte

FastRDF

C++ Prototyp aus ARVIDA

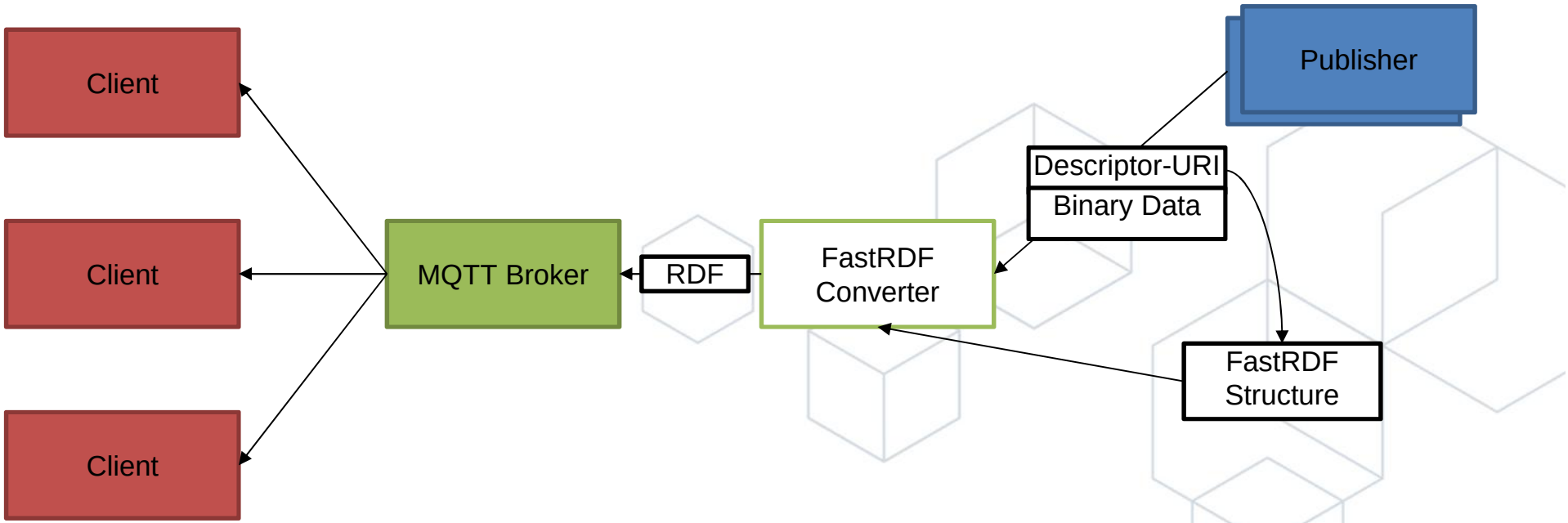
- Erweitert In-Memory Triplestore um direkte Callbacks zu nativen Daten
- Optimierter RDF N-Triple Serialisierer
- Unterstützt partielle Updates
- Reduktion der Nutzdaten-Serialisierung
  auf einfache Kopien von Speicherbereichen
- Performanz-Steigerung in
  Trackingszenario um mehr
  als eine Größenordnung

- Github:
  https://github.com/chrvog/FastRDF



Ausblick: FastRDF for IOT/WOT

- Nutzung eines MQTT Brokers zur Bereitstellung von RDF Sensor Daten
- Constrained IOT Devices senden Binärdaten
- FastRDF Micro-Service zur semantischen Anreicherung der Binärdaten



Vielen Dank!