ECONSTOR
Make Your Publications Visible.

A Service of

ZBW Leibniz-Informationszentrum Wirtschaft
Leibniz Information Centre for Economics

Clees, Tanja; Bareev-Rudy, Michael; Pfennig, Malte

Article — Published Version

Effizienter Datenmanagement-Workflow zur Analyse eines PtGtX-Systems basierend auf Open-Source-Software und Low-Cost-Hardware

HMD Praxis der Wirtschaftsinformatik

Provided in Cooperation with:
Springer Nature

Suggested Citation: Clees, Tanja; Bareev-Rudy, Michael; Pfennig, Malte (2024) : Effizienter Datenmanagement-Workflow zur Analyse eines PtGtX-Systems basierend auf Open-Source-Software und Low-Cost-Hardware, HMD Praxis der Wirtschaftsinformatik, ISSN 2198-2775, Springer Fachmedien Wiesbaden GmbH, Wiesbaden, Vol. 61, Iss. 4, pp. 891-910, https://doi.org/10.1365/s40702-024-01081-4

This Version is available at:
https://hdl.handle.net/10419/315893

Standard-Nutzungsbedingungen:

Die Dokumente auf EconStor dürfen zu eigenen wissenschaftlichen Zwecken und zum Privatgebrauch gespeichert und kopiert werden.

Sie dürfen die Dokumente nicht für öffentliche oder kommerzielle Zwecke vervielfältigen, öffentlich ausstellen, öffentlich zugänglich machen, vertreiben oder anderweitig nutzen.

Sofern die Verfasser die Dokumente unter Open-Content-Lizenzen (insbesondere CC-Lizenzen) zur Verfügung gestellt haben sollten, gelten abweichend von diesen Nutzungsbedingungen die in der dort genannten Lizenz gewährten Nutzungsrechte.

Terms of use:

Documents in EconStor may be saved and copied for your personal and scholarly purposes.

You are not to copy documents for public or commercial purposes, to exhibit the documents publicly, to make them publicly available on the internet, or to distribute or otherwise use the documents in public.

If the documents have been made available under an Open Content Licence (especially Creative Commons Licences), you may exercise further usage rights as specified in the indicated licence.

http://creativecommons.org/licenses/by/4.0/deed.de

WWW.ECONSTOR.EU

Mitglied der Leibniz-Gemeinschaft

HMD Praxis der Wirtschaftsinformatik (2024) 61:891–910
https://doi.org/10.1365/s40702-024-01081-4

SCHWERPUNKT

Effizienter Datenmanagement-Workflow zur Analyse eines PtGtX-Systems basierend auf Open-Source-Software und Low-Cost-Hardware

Tanja Clees · Michael Bareev-Rudy · Malte Pfennig

Eingegangen: 17. Januar 2024 / Angenommen: 26. April 2024 / Online publiziert: 11. Juni 2024
© The Author(s) 2024

Zusammenfassung Um ein Power-to-Gas-to-X-System effizient zu optimieren, kann ein digitaler Zwilling als Simulationsmodell auf Basis experimenteller Daten für ein Laborsystem erstellt und entsprechend verändert werden. Darüber hinaus müssen für die Überwachung des realen Systems bzw. die Online-Simulation kontinuierlich Daten aus Experiment und Simulation erfasst und verarbeitet werden. Insgesamt ist ein effizienter Datenmanagement-Workflow erforderlich.

In dieser Arbeit wird ein Workflow aus freier, etablierter und skalierbarer Open-Source-Software für die vorliegende Anwendung skizziert und insbesondere ein geeignetes Datenmodell entwickelt, implementiert und seine ressourcensparende Realisierung auf kostengünstiger Hardware gezeigt. Abhängig von der Datenmodellierung kann preiswerte und alte Hardware für die geforderte Aufgabe ausreichend sein.

Mit Apache NiFi wird ein visueller Workflow zum Abrufen und Verarbeiten von Daten aus verschiedenen Quellen geschaffen. Die extrahierten Daten werden in Apache Cassandra aggregiert, einem Datensystem, das aufgrund seiner Leistung, Skalierbarkeit und Haltbarkeit häufig verwendet wird.

Grafana wird zur visuellen Überwachung des Systems eingesetzt. Das gesamte System wird mit Hilfe von Docker-Containern aufgebaut zum Zwecke der Reproduzierbarkeit und effizienten Bereitstellung.

Benchmarks und realistische Hardware- und Datenmodellierungskonfigurationen demonstrieren die Leistung der vorgeschlagenen Lösung.

✉ Tanja Clees · Michael Bareev-Rudy · Malte Pfennig
Institut für Technik, Ressourcenschonung und Energieeffizienz (TREE), Hochschule Bonn-Rhein-Sieg, Grantham-Allee 20, 53757 Sankt Augustin, Deutschland
E-Mail: tanja.clees@h-brs.de

Tanja Clees
Fraunhofer Institut für Algorithmen und Wissenschaftliches Rechnen SCAI, Schloss Birlinghoven, 53757 Sankt Augustin, Deutschland

Schlüsselwörter Power-to-Gas-to-X Mess- und Simulationsdaten · Open-source-basierter Datenworkflow · Datenmodellierung · Ressourcenschonender Hardwareaufbau

Efficient Data Management Workflow for Analysis of a PtGtX System Based on Open-Source Software and Low-cost Hardware

Abstract In order to efficiently optimize a power-to-gas-to-X system, a digital twin can be created as a simulation model based on experimental data for a laboratory system and modified accordingly. In addition, data from experiment and simulation must be continuously recorded and processed for monitoring and online simulation. Overall, an efficient data management workflow is required.

In this paper, a workflow consisting of free, established and scalable open-source software is outlined for the application at hand and, in particular, a suitable data model is developed, implemented and its resource-saving realization on low-cost hardware is demonstrated. Depending on the data modeling, inexpensive and old hardware may be sufficient for the required task.

Apache NiFi is used to create a visual workflow for retrieving and processing data from various sources. The extracted data is aggregated in Apache Cassandra, a data system widely used for its performance, scalability and durability.

Grafana is used for visual monitoring of the system. The entire system is built using Docker containers for the purpose of reproducibility and efficient deployment.

Benchmarks and realistic hardware and data modeling configurations demonstrate the stated performance and sustainability of the proposed solution.

Keywords Power-to-gas-to-X measurement and simulation data · Open-source-based data workflow · Data modeling · Resource-saving hardware design

1 Einleitung

Um ein Power-to-Gas-to-X-System (PtGtX) für unterschiedliche Anwendungsfälle effizient zu optimieren (Bareev-Rudy et al. 2023), kann ein Simulationsmodell für ein bestimmtes existierendes (Labor-)System erstellt und dann entsprechend verändert werden. Auf der Grundlage experimenteller Daten aus dem Labor werden verschiedene Parameter der Komponenten des Simulationsmodells auf ihre physischen Gegenstücke kalibriert, um einen konsistenten digitalen Zwilling für diese spezifische Situation zu erstellen. Darüber hinaus müssen für die Überwachung des realen Systems oder sogar für die Online-Simulation kontinuierlich Daten aus Experiment und Simulation performant und robust erfasst und verarbeitet werden. Insgesamt ist ein effizienter Datenmanagement-Workflow erforderlich.

Zur schnellen Übertragung und Auswertung von großen Datenmengen wurde der sog. SMACK Stack (Spark, Mesos, Akka, Cassandra, Kafka) vorgeschlagen, siehe z. B. Estrada (2016) und Cologne Intelligence (2019). Workflows für Anwendungen im Energiebereich und/oder für Simulationsdaten werden seit einigen

Jahren diskutiert. Gomez (2018) betrachtet das Archivieren von Wetter- und Smart-Meter-Daten und die Weiterverarbeitung mit Machine Learning. Die quelloffenen Software-Tools Apache Kafka (https://kafka.apache.org/), Apache NiFi (https://nifi.apache.org) und Apache Cassandra (https://cassandra.apache.org) werden verwendet. Hagenmeyer (2019) betrachtet im sog. Process Operation Framework (PROOF) Kafka und NiFi und listet mehrere NOSQL-Datensysteme v. a. für Anwendungen in der Energienetz-Simulation. Liu (2023) setzt darauf auf, begründet die Nutzung von Docker (https://www.docker.com/) und betrachtet weitere Simulationsanwendungen im Energiebereich. Monsberger (2020) schlägt für Energiemanagement von Gebäuden einen Workflow mit MQTT (https://mqtt.org/), NiFi und einer PostgreSQL-Datenbank (https://www.postgresql.org/) vor.

Entscheidend für die Leistung in Bezug auf Durchsatz und Speicherplatzbedarf ist die Datenmodellierung, d. h. die Berücksichtigung von Inhalt, Größe und Häufigkeit der Daten. Abhängig von diesen Faktoren kann preiswerte und alte Hardware für die geforderte Speicheraufgabe ausreichend sein. In den oben genannten Arbeiten wird zumeist das konkrete Datenmodell nicht betrachtet bzw. seine Leistungsfähigkeit für die jeweilige Anwendung nicht diskutiert.

In dieser Arbeit wird ein Workflow aus freier, etablierter und skalierbarer Open-Source-Software für die vorliegende Anwendung skizziert und insbesondere ein geeignetes Datenmodell entwickelt, implementiert und seine ressourcensparende Realisierung auf kostengünstiger Hardware gezeigt. Dabei wird konkret ausrangierte Hardware genutzt, um sie im Sinne der Nachhaltigkeit einer weiteren, sinnvollen Verwendung zuzuführen. Die Datenmodellierung orientiert sich an realistischen Systemen, wie sie beispielsweise kommerziell für autarke Versorgung mit Energie aus erneuerbaren Quellen von GKN Hydrogen (https://www.gknhydrogen.com/hydrogen-offgrid/) oder Wilo (https://wilo.com/de/Pioniergeist/Wasserstoff/) bzw. von Heliocentris (https://www.heliocentrisacademia.com/new-energy-lab/p1526) für das Labor angeboten werden. Diese Systeme können als Ersatz für konventionelle, mit fossilen Brennstoffen arbeitenden Blockheizkraftwerken bzw. Notstromaggregaten angesehen werden (Clees et al 2024).

Experimentelle Daten werden oft von proprietären Systemen erstellt. In der Regel kann auf die Daten über Standardprotokolle wie FTP, HTTP oder MQTT zugegriffen werden, oder sie können in für Menschen lesbare Formate wie CSV (Shafranovich 2005), XML (Bray 2008) oder JSON (Bray 2017) exportiert werden. Apache NiFi mit seinen zahlreichen Schnittstellen und Verarbeitungsmöglichkeiten wird verwendet, um einen visuellen und überschaubaren Workflow zum Abrufen und (Vor-)Verarbeiten von Daten aus verschiedenen Quellen zu schaffen. Der Hauptvorteil ist, dass Daten aus verschiedenen Quellen und Formaten zentralisiert und über eine Schnittstelle mit demselben Ausgabeformat zugänglich sind. Die extrahierten Daten werden in Datensystemen wie dem etablierten Apache Cassandra aggregiert, einem Datensystem, das aufgrund seiner Leistung, Skalierbarkeit und Haltbarkeit häufig verwendet wird.

Schließlich wird Grafana (https://grafana.com/) verwendet, um die in Cassandra gespeicherten Daten zu visualisieren und ein System in Echtzeit zu überwachen. Das gesamte System wird mit Hilfe von Docker-Software-Containern aufgebaut.

Dies ermöglicht die Reproduzierbarkeit des Arbeitsablaufs auf einer Vielzahl von Hosts und damit eine effiziente Bereitstellung.

Benchmarks für ein spezifisches Laborsystem (Heliocentris Academia 2020) an der Hochschule Bonn-Rhein-Sieg und realistische Hardware- und Datenmodellierungskonfigurationen demonstrieren die angegebene Leistung und Nachhaltigkeit der vorgeschlagenen Lösung.

Der Beitrag ist wie folgt gegliedert: Abschn. 2 beschreibt den Daten-Workflow und das Deployment. Abschn. 3 diskutiert die Ergebnisse, insbesondere auch für die PtGtX-Anlage im Labor. Abschn. 4 fasst die wichtigsten Erkenntnisse zusammen und gibt einen Ausblick auf zukünftige Arbeiten.

2 Aufbau des Workflows

2.1 Überblick

Der Aufbau des Daten-Workflows orientiert sich an der Lambda-Architektur, welche verwendet wird, um Daten im Big-Data-Kontext auf robuste, fehlertolerante, schnelle, skalierbare und einfach zu verwaltende Weise zu verarbeiten. Die Struktur besteht idealtypisch aus einem Serving-, Speed- und Batch-Layer, wobei letztere zwei die jeweils nötigen Rohdaten aus dem Masterdatensatz ziehen (Marz and Warren 2015).

Die Lambda-Architektur ist im Kontext von sozialen Netzwerken entstanden, die viele unsortierte Daten von deren Webseiten bekommen. Allerdings bietet sich die Struktur auch an, um Maschinen-Messdaten zu verarbeiten. Im vorliegenden Paper liegt der Fokus darauf, wie Messdaten in den Masterdatensatz gelangen, der in einer Laborumgebung für alle Mitarbeitenden zur Verfügung steht. Der Masterdatensatz bietet eine Grundlage, um die Datenverarbeitung bis hin zu einem digitalen Zwilling aufzubauen.

Die Lambda-Architektur befindet sich in der Regel in der Cloud. In der Edge befinden sich IoT-Geräte, die mit der Umgebung interagieren. Um Datenübertragungen zwischen IoT-Geräten und dem Masterdatensatz, der sich in der Cloud befindet, herzustellen, werden im Folgenden basierend auf Bareev-Rudy (2020) und Pfennig

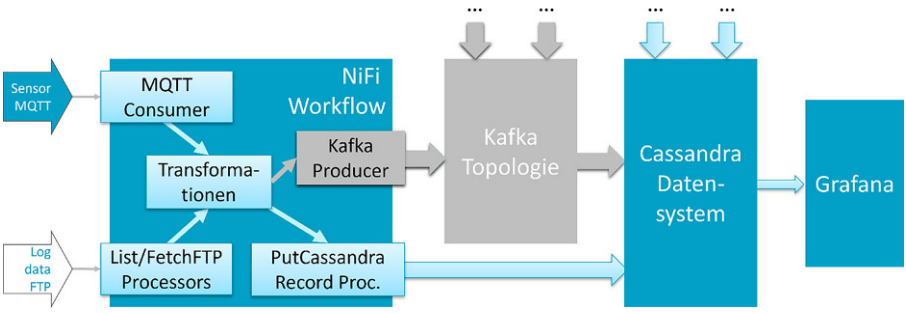

Abb. 1 Übersicht des betrachteten Gesamt-Workflows. Die oberen Eingänge symbolisieren mögliche weitere Datenquellen. Grau hinterlegt ist eine Erweiterung für größere Systeme

(2021) die genutzten Programme und eine mögliche Erweiterung vorgestellt. Eine Skizze des hier verwendeten Daten-Workflows ist in Abb. 1 dargestellt.

2.1.1 MQTT

Message Queuing Telemetry Transport (MQTT, https://mqtt.org/) ist ein Nachrichten-Protokoll für die „machine to machine“ (M2M) Kommunikation. Es basiert auf dem Publish-Subscribe-Prinzip und eignet sich für ein großes Netzwerk aus vielen kleinen IoT-Geräten, die über einen Back-End-Server überwacht werden. Das Nachrichten-Protokoll bietet nur wenig Einstellungsmöglichkeiten (Naik 2017). Open-Source-Software wie Eclipse Mosquitto (Broker, https://mosquitto.org/) bzw. die Eclipse Paho Bibliothek (Client, https://eclipse.dev/paho/index.php) bieten Implementierungen. Nachrichten werden als Binärcode gesendet, was Datentypen unterschiedlichster Art erlaubt (ISO/IEC 20922 2016). Hauptsächlich wird das Transmission Control Protocol (TCP) verwendet. Die Nachrichten lassen sich mithilfe von SSL/TLS verschlüsseln.

2.1.2 Apache Kafka

Apache Kafka ist eine Event-Streaming Software, die auf dem Publish-Subscribe-Mechanismus basiert (Koshy 2016, https://kafka.apache.org/). Kafka wurde konzipiert als verteilbares, skalierbares, dauerhaftes und fehlertolerantes Nachrichten-System mit hohem Datendurchsatz (Wu et al. 2019). Ankommende Nachrichten werden in Form von Logs gespeichert und zur Datensicherung auf andere Server repliziert, anders als bei MQTT. Konsumenten können diese Nachrichten in nahezu Echtzeit oder zu einem späteren Zeitpunkt lesen.

Kafka kann sowohl als Datensystem für den Masterdatensatz als auch als Datenverteilsystem innerhalb der Cloud-Umgebung genutzt werden. Kafka bietet anders als MQTT keine geeignete Möglichkeit, um kleine IoT-Geräte anzubinden. Für Datenaustausch zwischen Kafka und MQTT wird ein Konnektor benötigt, der aktuell nicht lizenzfrei verfügbar ist.

2.1.3 Apache NiFi

Eine Alternative für die Datenübertragung zwischen der Cloud und der Edge ist Apache NiFi (https://nifi.apache.org). Mit Hilfe seiner webbasierten Schnittstelle lassen sich Daten-Workflows erstellen und über viele gängige Schnittstellen zwischen Systemen auszutauschen. Dazu werden sogenannte *Consumers, Producers* und *Prozessoren* miteinander verbunden. Daten werden in Form von Nachrichten (*FlowFiles*) ausgetauscht. Die *Connections* zwischen den Prozessoren ermöglichen das Queuing und Puffern. Ein Beispiel findet sich in Abb. 2.

2.1.4 Apache Cassandra

Apache Cassandra (https://cassandra.apache.org) kann große Datenmengen verarbeiten und sie auf verschiedene Knoten verteilen und replizieren. Cassandra ist für hohe

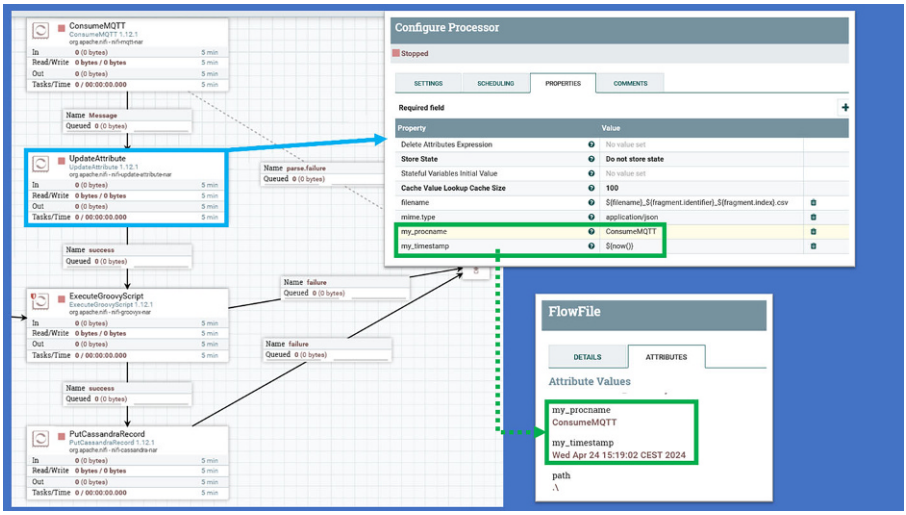

Abb. 2 *Linke Seite*: Einfacher NiFi-Workflow zum Bezug von Daten über MQTT (§ 2.1.1, ConsumeMQTT-Producer), Ergänzung von Attributen (vgl. auch § 2.2.3, UpdateAttribute-Prozessor), Manipulation mittels Groovy (vgl. auch § 2.2.3, ExecuteGroovyScript-Prozessor) und Speicherung in Cassandra (§ 2.2, PutCassandraRecord-Consumer). Die Producer, Prozessoren, Consumer werden mit Connections verbunden (siehe gezeigte Beispiele für „success“, „parse.failure“ und „failure“). *Rechte Seite*: Beispielhafte Konfiguration von UpdateAttribute mit ergänzten Attributen (hier: my_procname, my_timestamp) und resultierenden Werten dieser Attribute in einem erzeugten FlowFile

Verfügbarkeit und Fehlertoleranz ausgelegt und erlaubt eine Entscheidung zwischen hoher Datenkonsistenz und Übertragungsgeschwindigkeit.

Cassandra besteht aus Keyspaces, die Tabellen mit Daten enthalten. Keyspaces haben einen Replikationsfaktor und eine Replikationsstrategie (DataStax 2020a). Die Tabellen bestehen aus Spalten, die jeweils einen Namen, einen Datentyp und einen Wert haben. Für die Partitionierung und Sortierung (Clustering pro Partition) der Daten wird ein *Primary Key* verwendet (DataStax 2020b). Dieser besteht entsprechend aus einem *Partition Key* und einem *Clustering Key*.

Datenmodellierung mit Cassandra wird in § 2.3 beschrieben, das hier verwendete Speicherformat in (Apache Software Foundation 2019). Wichtige Stichworte sind *delta encoding* und die *dense/sparse*-Schätzung.

2.1.5 Zeitsynchronisation zwischen Nodes

Damit dezentrale Systeme miteinander arbeiten können, ist eine Zeitsynchronisation unerlässlich (Rinaldi et al. 2017). Besonders bei älterer Hardware konnten die Autor:innen feststellen, dass die Real Time Clock (RTC) auf dem Motherboard nicht mehr gut funktioniert. Dadurch entsteht ein Zeitunterschied zwischen einzelnen Rechnern (Zeitdrift). Um dies zu verhindern, wird die Zeit mit dem Network Time Protocol (NTP) synchronisiert. Beim NTP können Genauigkeit von 100 ms erreicht werden. Für eine höhere Genauigkeit sollte das Precision Time Protocol

(PTP) verwendet werden. Das Simple Network Time Protocol (SNTP) ist für Serveranwendungen nicht empfehlenswert (Kerö et al. 2019).

2.2 Übertragung der Daten von MQTT/FTP über NiFi an Cassandra

Betrachtet werden hier typische Varianten des Mess- bzw. Simulationsdatenbezugs:

- *ConsumeMQTT* für von IoT-Geräten übermittelte Sensordaten
- *ListFTP* und *FetchFTP* für Dateien von FTP-Servern

Über NiFis *PutCassandraRecord* werden die FlowFile-Daten an Cassandra gesendet. PutCassandraRecord kann verschiedene Formate verarbeiten. Hier wird JSON verwendet. Dadurch wird sichergestellt, dass die Daten mit den richtigen Datentypen interpretiert werden, wodurch eine typische Fehlerursache beseitigt wird. Bemerkungen zu CSV bzw. XML werden in § 2.2.2 gemacht.

2.2.1 Bezug von Daten über MQTT bzw. FTP

Der Bezug von Daten über MQTT wird über NiFis ConsumeMQTT direkt unterstützt. Das Übertragen von Daten über FTP gestaltet sich aufwändiger. NiFis *GetFTP* kann zwar eine Verbindung zu einem FTP-Server herstellen und den Inhalt von Dateien in FlowFiles ablegen. Ein passenderer Weg ist aber folgender: *ListFTP* durchsucht das Verzeichnis nach Dateien. Nur wenn eine neue Datei gefunden wird, erstellt ListFTP ein FlowFile. Dieses wird an *FetchFTP* weitergeleitet, der den Inhalt der Datei zum FlowFile hinzufügt. ListFTP kann Dateien nach Zeitstempel oder Entität verfolgen. Entität bedeutet hier, dass jede einzelne Datei anhand ihrer Metadaten verfolgt wird; das braucht aber mehr Ressourcen als die Verfolgung nach Zeitstempel.

2.2.2 Bemerkungen zur NiFi-Vorverarbeitung von CSV- und XML-Dateien

In NiFi ist eine funktionsreiche Laufzeitumgebung für Groovy (https://groovy-lang.org/) integriert, welche der Erfahrung der Autor:innen nach der NiFi internen Python-Laufzeitumgebung überlegen ist. NiFis *ExecuteGroovyScript* kann ein Groovy-Skript auführen. Dieses kann dann z. B. im Format korrekte CSV- oder JSON-Strings erstellen oder auch Datenauszüge aus XML bzw. numerische Umrechnungen (z. B. in eine andere physikalische Einheit bzw. ein anderes Zeitformat) vornehmen.

PutCassandraRecord kann zwar auch CSV-Daten verarbeiten, erzeugt aber Fehler, da einige Dezimalzahlen als Strings interpretiert werden. Daher wird *ConvertRecord* verwendet, der CSV in JSON konvertiert, wobei die Daten in den Spalten explizit als Zahl, String oder Zeitstempel definiert werden. *SplitCSVLines* teilt eingehende FlowFiles in FlowFiles mit bis zu 100 Zeilen auf. Dies ist die Höchstgrenze, die PutCassandraRecord gleichzeitig in Cassandra speichern kann.

2.2.3 NiFi Error Handling

Auch bei gut konzipierten Systemen braucht es eine Strategie zur Fehlerbehandlung. Zwei verschiedene Arten von Fehlern werden hier betrachtet:

- Aufgrund von Verbindungsunterbrechungen konnten Daten nicht abgerufen oder gespeichert werden.
- Die (eigentlich korrekten) Daten sind nicht korrekt angekommen und können daher nicht verarbeitet werden.

Beide Fehlerarten sind (i. d. R.) durch Wiederholung behebbar, allerdings auf technisch unterschiedlichem Weg. Für die erste Fehlerart wird der Pfad für fehlgeschlagene FlowFiles mit dem Eingang des Prozessors verbunden. Die zweite Fehlerart würde durch diese Maßnahme für immer in dieser Schleife „feststecken". *RetryFlowFile* wird stattdessen benutzt.

NiFi erstellt Fehlermeldungen mit Zeitstempeln für seine Prozessoren. Es ist hilfreich, Prozessor und Zeitpunkt des Fehlers zu kennen. *UpdateAttribute* kann beide Angaben als zusätzliche Attribute zum eingehenden FlowFile hinzufügen, das von einer Verbindung für fehlgeschlagene FlowFiles kommt (vgl. beispielhafte neue Attribute in Abb. 2). Wenn jedoch ein FlowFile NiFi verlässt, wird nur die Nutzlast (Inhalt) exportiert, während diese zusätzlichen Angaben verschwinden. Um dies zu verhindern, werden *AttributesToJSON* und *ReplaceText* verwendet. Anschließend kann das FlowFile z. B. auf einem FTP-Server gespeichert bzw. per E-Mail verschickt werden.

a

Spaltenname	Systemkomponente	Bedeutung	Datentyp	Größe [Bytes]
date	Clock	Datum des Messtages, zentriert um den 1.1.1970 (Unix Epoch)	date	4
time		Uhrzeit der Messung in Millisekunden seit Mitternacht des Messtages	int	4
bat_current	Batterie	Stromstärke	float	4
bat_soc		State-of-Charge (Ladezustand in %)	float	4
bat_temp		Temperatur	float	4
bat_voltage		Spannung	float	4
el_current	Last	Stromstärke	float	4
el_voltage		Spannung	float	4
fc_current	Brennstoffzelle	Stromstärke	float	4
fc_currentint		Stromstärke zwischen Stack und DC/DC-Wandler	float	4
fc_h2flow		Wasserstoff-Fluss	float	4
fc_h2press		Wasserstoff-Druck	float	4
fc_voltage		Spannung	float	4
fcc_current		Stromstärke	float	4
hg_current	Elektrolyseur	Stromstärke	float	4
hg_h2flow		Wasserstoff-Fluss	float	4
hg_h2press		Wasserstoff-Druck	float	4
id_temp	Innenraum	Temperatur	float	4
ipc_current	System	AC-Output-Stromstärke	float	4
isl_current	Insel-Grid	AC-Input-Stromstärke	float	4
isl_voltage		AC-Input-Spannung	float	4
mh1_temp	Metallhydridspeicher	Temperatur von Flasche 1	float	4
mh2_temp		Temperatur von Flasche 2	float	4
mh3_temp		Temperatur von Flasche 3	float	4
od_temp	Außenbereich	Temperatur	float	4
pg_current	Netzanschluss	AC-Input-Stromstärke	float	4
pg_voltage		AC-Input-Spannung	float	4

b

```
CREATE TABLE nel.measurement (
    date date,
    time int,
    bat_current float,
    bat_soc float,
    bat_temp float,
    bat_voltage float,
    el_current float,
    el_voltage float,
    fc_current float,
    fc_currentint float,
    fc_h2flow float,
    fc_h2press float,
    fc_voltage float,
    fcc_current float,
    hg_current float,
    hg_h2flow float,
    hg_h2press float,
    id_temp float,
    ipc_current float,
    isl_current float,
    isl_voltage float,
    mh1_temp float,
    mh2_temp float,
    mh3_temp float,
    od_temp float,
    pg_current float,
    pg_voltage float,
    PRIMARY KEY (date, time)
) WITH CLUSTERING ORDER BY (time ASC)
```

Abb. 3 **a** Messdaten mit Zuordnung zur jeweiligen Komponente des hier betrachteten PtGtX-Systems im Labor inkl. Bedeutung, Cassandra-Datentyp und zugehörge Größe in Bytes. **b** Erzeugung der entsprechenden Cassandra-Tabelle

2.3 Datenmodellierung

Da hier Cassandra verwendet wird, orientieren sich Aussagen zum Speicherbedarf und entsprechenden Optimierungsmöglichkeiten daran (Apache Software Foundation 2024; Carpenter und Hewitt 2020).

2.3.1 Typische Tabellenformate

Die vorliegende Anwendung braucht sowohl Messdaten (für ein Beispiel siehe Abb. 3) als auch Daten aus Simulationsläufen. Beide Arten speichern mit Zeitstempeln behaftete Werte. Gleichartige Tabellenformate werden im Folgenden vorgeschlagen und diskutiert. Der einzige Unterschied ist die Interpretation einer Spalte *scen_ID*, welche bei Messungen für die Kennzeichnung des physikalischen Systems, bei Simulationsläufen für das konkrete Simulationsszenario stehen möge.

Folgende Formate werden betrachet:

- $\boldsymbol{T_{breit}}$**:** primarykey(**(scen_ID, timestamp_part1)**, timestamp_part2)
  mit $N_m = N_{sensors}$ Spalten für die Messwerte
- $\boldsymbol{T_{breit,scen}}$**:** primarykey(**(timestamp_part1)**, timestamp_part2, scen_ID)
  mit $N_m = N_{sensors}$ Spalten für die Messwerte
- $\boldsymbol{T_{lang}}$**:** primarykey(**(scen_ID, sensor_ID, timestamp_part1)**, timestamp_part2)
  mit $N_m = 1$ Spalte für den Messwert zum jeweiligen Sensor mit der ID **sensor_ID**
- $\boldsymbol{T_{lang,scen}}$**:** primarykey(**(sensor_ID, timestamp_part1)**, timestamp_part2, scen_ID)
  mit $N_m = 1$ Spalte für den Messwert zum jeweiligen Sensor mit der ID **sensor_ID**

Hierbei bezeichnen *timestamp_part1* und *timestamp_part2* je ein oder mehrere Spalten, welche zusammen den Zeitstempel ergeben. Im Beispiel (Abb. 3) entspricht „date“ dem *timestamp_part1* und „time“ dem *timestamp_part2* Der Primary Key sei jeweils eindeutiger Identifizierer der Datenreihe. I.d.R. ist dies neben *scen_ID* (Anzahl $N_{scenarios}$) und *sensor_ID* (Anzahl $N_{sensors}$) durch die Auflösung des Zeitstempels entsprechend der Aufzeichnungsfrequenz erreicht. In der Praxis ergibt eine Verschiebung von (mindestens) sensor_ID in den Cluster Key i.d.R. zu große Partitionen und wird daher nicht betrachtet.

2.3.2 Bestimmung der Partitionsgröße

Tab. 1 definiert grundlegende Größen zur Bestimmung der Partitionsgröße.

Unter der Annahme konstanter Anzahlen (sonst als Maxima zu interpretieren), ist die Gesamtanzahl aller Spalten $N_a = N_p + N_s + N_c + N_m$.

Tab. 1 Größen für die Spalten einer Cassandra-Tabelle T (vgl. Carpenter und Hewitt 2020)

Spaltentyp	Anzahl	Datentypen
Partition Key	N_p	$c_{p,i}$
Statisch	N_s	$c_{s,i}$
Clustering	N_c	$c_{c,i}$
Weitere	N_m	$c_{m,i}$

N_r sei die (maximale) Anzahl der Reihen in der Partition. I.d.R. entspricht sie der Anzahl der „Log-Vorgänge“ von Messungen in der Partition. N_r wird als konstant über alle Partitionen angenommen. Dann ist die Anzahl N_{table} an Tabellenwerten in einer Partition:

$$N_{\text{table}} = N_r \left(N_a - N_p - N_s\right) + N_s = N_r \left(N_c + N_m\right) + N_s$$

Man beachte aber, dass die Anzahl N_v an Daten in einer Partition mit einem möglicherweise eigenen Zeitstempel sich wie folgt berechnet (anders als in Apache Software Foundation (2024) sowie Carpenter und Hewitt (2020) angegeben):

$$N_v = N_r \left(1 + N_m\right) + N_s$$

Die „1“ steht dabei für die aggregierten Clusterkeys, für die immer der gleiche Zeitstempel geschrieben werden muss. Er könnte zu den Zeitstempeln der Werte der weiteren Spalten verschieden sein (siehe auch § 3), auch wenn das in der Praxis bei den hier betrachteten Tabellen selten vorkommen wird.

Dann ist die Größe P_T einer Partition der Tabelle T unkomprimiert ca. max.

$$P_T = \sum_{i=1}^{N_p} \left|c_{p,i}\right| + \sum_{i=1}^{N_s} \left|c_{s,i}\right| + \boldsymbol{N_r} \left(\sum_{i=1}^{N_c} \left|c_{c,i}\right| + \sum_{i=1}^{N_m} \left|c_{m,i}\right| \right) + \boldsymbol{s_o N_v}$$

Dabei bezeichnet $|c_*|$ die Größe des Datentyps c_* in Bytes (z. B. 4 Bytes für float und int), $s_o = 8$ Bytes (vernünftige Arbeitsannahme).

In Cassandra muss gelten $N_v < 2$ Mrd.. Aus Performancegründen soll P_T unter 100 MBytes, besser viel kleiner sein. Die Autor:innen schlagen vor, unkomprimiert auf unter 100 Mbytes zu optimieren und die Tabellen mit Default-Kompression zu erstellen.

2.3.3 Bestimmung des Gesamtspeicherbedarfs

N_{parts} sei die Gesamtzahl der Partitionen (inklusive Replikation). Dann ist der Gesamtspeicherbedarf $S_T = N_{\text{parts}} * P_T$.

Nachfolgend wird der Einfachheit halber $N_s = 0$ angenommen. Die summierten Bytelängen der Partitions- bzw. Clusterkey-Elemente werden folgendermaßen abgekürzt:

$$B_p := \sum_{i=1}^{N_p} \left|c_{p,i}\right|, \quad B_c := \sum_{i=1}^{N_c} \left|c_{c,i}\right|, \quad B_m := \sum_{i=1}^{N_m} \left|c_{m,i}\right|$$

Dann ist

$$P_T = B_p + N_r \left(B_c + B_m\right) + s_o N_r \left(1 + N_m\right) = B_p + N_r \left(B_c + s_o + B_m + s_o N_m\right)$$

Unter der weiteren, für Messwerte realistischen Annahme, dass alle $|c_{m,i}| = b_m$ gleich sind (z. B. $b_m = 4$ bei float), erhält man $B_m = b_m N_m$ und damit

$$P_T = B_p + N_r \left(B_c + s_o + (b_m + s_o)\, N_m\right)$$

2.3.4 Vergleich zweier typischer Tabellenformate

Die Formate $\boldsymbol{T_{breit}}$ und $\boldsymbol{T_{lang}}$ werden nachfolgend verglichen. Es gilt:

$$N_r\,(\text{Tlang}) = N_r\,(\text{Tbreit}) \quad \text{und} \quad N_{\text{parts}}\,(\text{Tlang}) = N_{\text{sensors}} * N_{\text{parts}}\,(\text{Tbreit})$$

Dann sind

$$P_T(\text{Tbreit}) = B_p(\text{Tbreit}) + N_r \left(B_c + s_o + (b_m + s_o) * N_{\text{sensors}}\right)$$
$$P_T\,(\text{Tlang}) = B_p\,(\text{Tlang}) + N_r \left(B_c + s_o + (b_m + s_o) * 1\right)$$

Und somit

$$S_T\,(\text{Tbreit}) =$$
$$N_{\text{parts}}\,(\text{Tbreit}) * \left(B_p\,(\text{Tbreit}) + N_r \left(B_c + s_o + (b_m + s_o) * N_{\text{sensors}}\right)\right)$$
$$S_T\,(\text{Tlang}) =$$
$$N_{\text{sensors}} * N_{\text{parts}}\,(\text{Tbreit}) * \left(B_p\,(\text{Tlang}) + N_r \left(B_c + s_o + (b_m + s_o) * 1\right)\right)$$
$$= N_{\text{parts}}\,(\text{Tbreit}) * \left(N_{\text{sensors}} B_p\,(\text{Tlang}) + N_r N_{\text{sensors}} \left(B_c + s_o + (b_m + s_o) * 1\right)\right)$$

Für typische Werte wie $s_o = 8$ [Bytes] und $B_c = b_m = 4$ [Bytes] erhält man

$$S_T\,(\text{Tbreit}) = N_{\text{parts}}\,(\text{Tbreit}) * \left(B_p\,(\text{Tbreit}) + 12 N_r\,(1 + N_{\text{sensors}})\right)$$
$$S_T\,(\text{Tlang}) = N_{\text{parts}}\,(\text{Tbreit}) * \left(N_{\text{sensors}} B_p\,(\text{Tlang}) + 24\, N_r N_{\text{sensors}}\right)$$

Weil B_p(Tbreit) bzw. $N_{sensors}B_p$(Tlang) im Vergleich zum Rest kleine Zahlen sind, ergibt sich also

$$S_T\,(\text{Tlang}) \approx 2\, \frac{N_{\text{sensors}}}{N_{\text{sensors}} + 1}\, S_T\,(\text{Tbreit})$$

Damit braucht für hinreichend große $N_{sensors}$ $\boldsymbol{T_{lang}}$ fast zweimal so viel Speicher wie $\boldsymbol{T_{kurz}}$.

2.3.5 Materialized Views

Cassandra erlaubt das Anlegen von abgeleiteten Tabellen, sog. *Materialized Views*. Diese werden nicht durch Indexierung, sondern durch Speicherung von Partitionen wie bei normalen Tabellen realisiert. Vorteile sind die automatische Pflege und der schnelle Zugriff. Nachteil ist der nötige Plattenplatz.

2.4 Visuelle Überwachung mit Grafana

Schließlich wird die Open-Source-Software Grafana (https://grafana.com/) verwendet, um die in Cassandra gespeicherten Daten zu visualisieren und ein System in (Quasi-)Echtzeit zu überwachen.

Grafana kann mit vielen gängigen Datenbanken über offizielle oder Community-Plugins verbunden werden. Es kann Daten in verschiedenen Diagrammen und interaktiven Graphen anzeigen. Da Grafana über eine Weboberfläche konfiguriert und verwendet wird, kann es vielen Benutzern zugänglich gemacht werden, mit individuell definierbaren Rollen und Berechtigungen.

Das Plugin für Cassandra setzt für möglichst effiziente Abfragen eine Tabellenstruktur voraus. Alternativ kann ein eigenes Backend geschrieben werden, welches mit dem Grafana JSON Plugin interagiert. Die Autor:innen haben dafür das Backend *CasAPI* mit Python erstellt. Die Bibliothek DataStax cassandra-driver stellt die Verbindung zu Cassandra her, während das Framework Flask (https://flask.palletsprojects.com/en/3.0.x/) verwendet wurde, um ein HTTP-Backend zu erstellen.

2.5 Deployment

Das gesamte System wird mit Hilfe von Docker-Containern aufgebaut. Dies ermöglicht die Reproduzierbarkeit des Arbeitsablaufs auf einer Vielzahl von Hosts und damit eine effiziente Bereitstellung.

2.5.1 Docker

Docker (https://www.docker.com/) ist ein quelloffenes, plattformübergreifendes Tool zur Bündelung von Anwendungen und deren Ausführung in Containern. Es ist weit verbreitet (Kinsta 2023).

Ein Container ist, einfach ausgedrückt, eine Umgebung, die auf dem Host-System läuft, aber von diesem isoliert ist. Er kann mehrere Anwendungen und die erforderlichen Abhängigkeiten enthalten. Ein Vorteil besteht darin, dass im Vergleich zu einer Installation auf dem Hostsystem die Abhängigkeiten für die isolierte Anwendung keine Kompatibilitätsprobleme verursachen. Ein weiterer Vorteil ist die höhere Reproduzierbarkeit auf vielen Plattformen.

Docker-Container können ein gemeinsames virtuelles Netzwerk verwenden, um über TCP/IP miteinander zu kommunizieren. Da diese Container vom Host isoliert sind, kann eine Verbindung von außen nicht hergestellt werden. Um dies zu ermöglichen, müssen Ports von den Containern weitergeleitet werden.

In vielen Fällen ist es praktisch oder notwendig, auf Dateien innerhalb eines Containers zuzugreifen. Dies kann durch das Einbinden eines Datenträgers (Ordners) vom Host-System in einen Ordner innerhalb des Containers ermöglicht werden, wodurch ein „gemeinsamer“ Ordner zwischen Host und Container entsteht.

Um Container zu konfigurieren, Ports weiterzuleiten und Volumes einzuhängen, kann das zusätzliche Tool Docker-Compose verwendet werden.

2.5.2 *Datenerfassung- und aggregation auf Messrechner*

Das hier betrachtete PtGtX-System im Wasserstofflabor verfügt über einen integrierten Messrechner, auf dem die Steuerungssoftware für das System läuft. Über diese lassen sich die Messdaten als CSV ausgeben oder mithilfe einer API abrufen. Die API ist sehr rudimentär und unterstützt das Abfragen und Setzen von Werten und Zuständen über UDP. Um die Datenerfassung zu automatisieren, wurde ein Python-Skript geschrieben, welches alle Messwerte über UDP abfragt, mit einem gemeinsamen Zeitstempel versieht und als JSON über MQTT an NiFi sendet. Die Messoftware und somit auch die API aktualisieren die Messwerte „nur“ jede Sekunde, deshalb sind die benötigten 2 ms Abfragezeit sequenzieller UDP-Abfragen für alle Messwerte vernachlässigbar.

In NiFi wird mithilfe eines Groovy-Skripts das Datum aus dem Zeitstempel extrahiert, in das von Cassandra erwartete binäre Format gebracht und zur JSON hinzugefügt, woraufhin die Daten mit PutCassandraRecord in Cassandra geschrieben werden.

2.5.3 *Zeitsynchronisation*

Der Messrechner ist aus Sicherheitsgründen getrennt vom Hochschulnetz. Ohne Internetanschluss fängt die Uhrzeit des Messrechners an zu driften. Der Messrechner ist daher an den Server angeschlossen, welcher wiederum ans Internet angebunden ist. Aus diesem Grund läuft auf dem Server ein weiterer Docker Container mit *Chrony*, einem einfachen Zeitserver.

2.5.4 *Docker-Umsetzung für das Wasserstofflabor*

Um alle in den vorangegangenen Abschnitten beschriebenen Funktionen und Merkmale zu ermöglichen, werden Docker-Container verwendet (James 2019) und mit Docker-Compose (https://docs.docker.com/compose/) u. a. passende Ports konfiguriert. Die folgenden Container werden benutzt:

- Cassandra
- NiFi
- Mosquitto
- Grafana
- Chrony
- CasApi

Da sie sich im selben virtuellen Netzwerk befinden, wird allen Containern für eine vereinfachte Kommunikation eine IP-Adresse zugewiesen. Innerhalb des Netzwerks können sie sich gegenseitig über ihre Containernamen anstelle ihrer IP-Adressen ansprechen. Dies wird z. B. für die Konfiguration von PutCassandraRecord verwendet.

Für NiFi, Cassandra und Grafana werden sogenannte *Volumes* eingehängt, um eine dauerhafte Speicherung zu ermöglichen. Ohne solche Volumes würde sie jedes Herunterfahren mit Docker Compose in ihren Standardzustand zurücksetzen. Da ihr Speicherort nicht leicht zugänglich ist, wurden manuelle Einhängevorgänge ver-

wendet, um z. B. die Synchronisierung zwischen Systemen zu Sicherungszwecken zu ermöglichen.

2.5.5 Hardware des Labors

Der Server ist ein ehemals im PC-Pool der Hochschule stehender Computer mit einem Intel i5-3550 (4) @ 3,700 GHz CPU, 16 GB Arbeitsspeicher, 500 GB SSD Systemplatte, 4 TB Datenplatte und einer externen Netzwerkkarte für zwei Ethernetanschlüsse à 1 Gbit/s. Das verwendete Betriebssystem ist Ubuntu.

Der Messrechner des Wasserstofflabors läuft auf Windows 10, beinhaltet einen Intel Pentium Gold G5400 (2) @ 3,70 GHz CPU, 8 GB Arbeitsspeicher und eine 500 GB SSD Systemplatte. Er beinhaltet ebenfalls eine externe Netzwerkkarte und somit zwei Ethernetanschlüsse à 1 Gbit/s für jeweils die systeminterne und -externe Kommunikation.

3 Ergebnisse

Im Folgenden finden sich die Tabellendefinitionen und Ergebnisse für mehrere Benchmarks:

- B1: Tabellen schrittweise füllen. Alle Tabellen sind vom Typ $\boldsymbol{T_{breit,scen}}$, um im Benchmark das schrittweise Updaten der Zeitstempel und den dadurch bedingten Overhead $s_o N_v$ zu testen. Analoge Ergebnisse werden für die anderen Tabellenformen erreicht.
- B2: Cluster Keys separat updaten. Eine Auswahl der Tabellen aus B1 wird benutzt.
- B3: Partitionsgrößen und Gesamtspeicherverbrauch für das Labor und verwandte PtGtX-Systeme als Blockheizkraftwerke oder Notstromaggregate. Die Tabellen entsprechen $\boldsymbol{T_{breit}}$ und $\boldsymbol{T_{lang}}$ und werden in § 3.2 beschrieben.

3.1 Benchmarks zur Schätzformel für die Partitionsgröße

3.1.1 Beispieltabellen für B1 und B2

Für die unteren vier Tabellen wird die Kompression ausgeschaltet. Alle Spaltenbytelängen c_* betragen 4 Bytes (int oder float). Insbesondere N_c wird variiert.*hms* enthält 13 Spalten:

- primary key((year, month, day), hour, minute, second, scen_ID)
- Weitere Spalten: cons1, ..., cons6

hms_MV ist ein Materialized View zu *hms*:

- primary key((scen_ID,year), month, day, hour, minute, second)
- absteigende Ordnung („DESC“) im Clustering

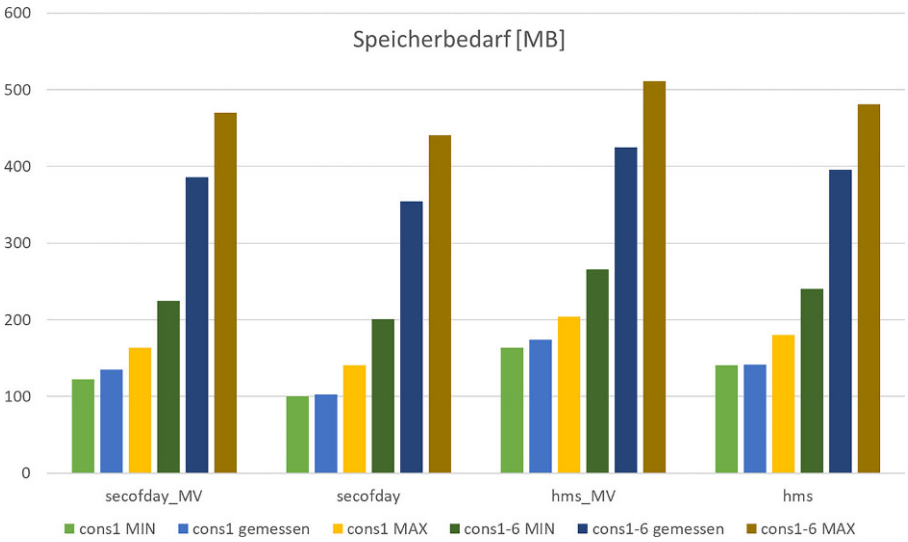

Abb. 4 Speicherbedarf gemessen und geschätzt. cons1(–6) MAX: $N_v = N_r\,(1 + N_m)$ mit $N_m = 1$ bzw. 6; cons1(–6) MIN: gleiche Zeitstempel aller Werte pro Clusterkey: $N_v = N_r$

secofday enthält 11 Spalten:

- primary key((year, month, day), sec_of_day, scen_ID)
- Weitere Spalten: cons1, ..., cons6

secofday_MV ist ein Materialized View zu *secofday*:

- primary key((scen_ID, year), month, day, sec_of_day)
- absteigende Ordnung („DESC“) im Clustering

3.1.2 Benchmark B1

Nach Anlegen der Tabellen werden schrittweise Daten immer nur für eine Wertespalte übertragen, also für den Primary Key und ein einzelnes cons1, ... cons6. Danach wird mit Hilfe von nodetool von Cassandra die Tabelle kompaktiert, bevor die Größe der data-Unterverzeichnisse summiert wird.

Die Ergebnisse sind erwartungsgemäß. Der Speicherbedarf wächst linear (ohne separate Abbildung hier, minimale und maximale Größen: siehe Abb. 4), die Materialized Views sind etwas größer als ihr Original. *secofday* spart gegenüber *hms* Platz ein.

Abb. 4 zeigt, dass die oberen Grenzen erwartungsgemäß nicht erreicht werden. Denn der MAX-Wert ist eine Schätzung für die Situation, in der alle Werte inklusive der Clusterkeys eigene Zeitstempel haben.

Tab. 2 Ergebnisse des Tests „Cluster Keys separat updaten“. „noCons geschätzt“: Hier ist $N_m = 0$, daher MIN-Wert gleich MAX-Wert

	noCons geschätzt [MB]	noCons gemessen [MB]	cons1 MIN geschätzt [MB]	Nur cons1 gemessen [MB]	noCons, dann cons1 gemessen [MB]	cons1 MAX geschätzt [MB]
Hms_MV	143	151	163	174	182	204
Hms	120	118	140	142	152	180

3.1.3 Benchmark B2

Für einen weiteren Test wird *hms* (und damit auch *hms_MV*) neu aufgesetzt. Zuerst werden nur Cluster-Key-Spalten, danach erst cons*-Werte gefüllt. Nach jedem Schritt wird kompaktiert.

Tab. 2 zeigt, wie die geschätzten Grenzen weitgehend erreicht bzw. überschritten werden:

- Es kann dauern, bis trotz Kompaktierung Cassandra die alten Werte ersetzt.
- s_o = **8 Bytes** ist nur eine Arbeitsannahme. Die Zeitstempel werden eigentlich in Mikrosekunden erstellt, dafür bräuchte es mehr Bytes. Cassandra arbeitet mit Differenzbildungen zu einem Referenzzeitstempel (der Partition, *delta encoding*) sowie Schätzungen zum Füllgrad der Zeilen. Daher schwankt der Bedarf.

3.2 Partitionsgrößen und Gesamtspeicherverbrauch – Benchmark B3

PtGtX-Systeme als Blockheizkraftwerke oder Notstromaggregate haben eine typische Größenordnung von $N_{sensors}$ im Bereich 20 bis 200 (teils aggregierten) Sensoren (Clees et al 2024). Es gibt jedoch auch Systeme mit sehr vielen kleinen Teilsystemen (beispielsweise sehr vielen kleinen Elektrolyseuren). Falls dort alle Sensordaten aufbewahrt werden sollen, kommt Tabellenform T_{breit} nicht mehr in Frage. Stattdessen kommt die Tabellenform T_{lang} zum Einsatz, weil sensor_ID dort im Partition Key zu finden ist.

Für das Labor können die verschiedenen Tabellenformen miteinander verglichen werden. Hier gilt $N_{\text{sensors}} = 25$ (siehe Abb. 3) und mindestens $N_{\text{scenarios}} = 250$ im Falle von Simulationen als Basis für eine optimale Auslegung (Bareev-Rudy et al. 2023) sowie der Bedingung einer zusätzlichen Ablage von Messdaten eines Systems. Replikation wird nicht verwendet. Daten über ein Mess- bzw. Simulationsjahr in sekündlicher Aufzeichnung werden gespeichert.

Tab. 3 Verschiedene Tabellen, Partitions- und Gesamtgrößen. Nr. 1–3 sind vom Typ T_{breit}, Nr. 4–6 vom Typ T_{lang}, Nr. 7 ein Materialized View dazu, Nr. 8 vom Typ $T_{lang,scen}$

	Primary Key	N_v	P_T [MB]	S_T [TB]
1	(scen_ID, year, month), day, hour, minute, second	6,96E+07	796,95	2,2801
2	(scen_ID, week_global), second_per_week	1,57E+07	179,96	2,274
3	(scen_ID, year, month, day), second_per_day	2,25E+06	25,708	2,2801
4	(scen_ID, sensor_ID, year, month), day, hour, minute, second	5,36E+06	61,304	4,3848
5	(scen_ID, sensor_ID, year, month, week), day 7, hour, minute, second	1,21E+06	13,843	4,373
6	(scen_ID, sensor_ID, week_global), day 7, hour, minute, second	1,21E+06	13,843	4,373
7	(scen_ID, week_global, day 7), hour, minute, second, sensor_ID	4,32E+06	55,618	4,9196
8	(sensor_ID, week_global, day 7), hour, minute, second, scen_ID	4,32E+07	556,18	4,9196

Tab. 3 zeigt, dass die Bedingungen aus § 2.3.2 für N_v erfüllt sind, für P_T erst bei hinreichend feiner zeitlicher Partitionierung und nicht für Typ $T_{lang,scen}$. Das Maximum dafür wäre hier $N_{\text{scenarios}} = 44$.

4 Zusammenfassung und Ausblick

Die geschätzte Gesamtspeichergröße S_T aus § 2.3.3 ist eine gute Approximation für den unkomprimierten Fall, wie durch Benchmarks 1 und 2 bestätigt wird. Man sieht insbesondere, dass bei Datenübertragungen, welche je die ganze Tabellenzeile schreiben, der Overhead begrenzt bleibt. In der betrachteten Anwendung bleibt man daher an der unteren Grenze der Schätzung, welche sich durch $N_v = N_r + N_s$ ergibt, muss aber umgekehrt Platz für Cassandras Aufräumarbeiten und die Kompression einplanen. Daher kann man die obere Grenze für den Overhead, die sich durch $N_v = N_r\,(1 + N_m) + N_s$ ergibt, sicherheitshalber zur Approximation nutzen.

Benchmark B3 zeigt dann, dass Tabellen vom Typ $\boldsymbol{T_{breit}}$ auf dem sehr günstigen Server des Labors Platz finden. Entsprechend § 2.3.4 werden die flexibleren Tabellen des Typs $\boldsymbol{T_{breit}}$ etwa doppelt so groß. Der Server war eigentlich ausrangiert und kann im Sinne der Nachhaltigkeit sinnvoll weiterverwendet werden.

$\boldsymbol{T_{lang(,scen)}}$ erlaubt zwar Materialized Views auch für die Sensoren, anders als $\boldsymbol{T_{breit}}$. Dies wird aber mit deutlich mehr Speicherbedarf erkauft. Bei lediglich einem Materialized View zu einem $\boldsymbol{T_{lang}}$ wird mehr als viermal so viel Speicher wie für das entsprechende $\boldsymbol{T_{breit}}$ gebraucht. $\boldsymbol{T_{breit}}$ wird daher im Labor eingesetzt. Bei Abfragen zu Zeitscheiben quer zu den Szenarios wird eine entsprechend schlechtere Performance in Kauf genommen, die im vorliegenden Fall nicht stört.

Die erarbeitete Datenmodellierung funktioniert nicht nur für die konkrete Laborumgebung, sondern auch für PtGtX-Systeme als Blockheizkraftwerke oder Notstromaggregate. Bei großer Anzahl von Sensoren kann auf das Datenmodell $\boldsymbol{T_{lang}}$ gewechselt werden, welches dann auch Materialized Views erlaubt.

In weiteren Untersuchungen sollen zeitliche Benchmarks sowie Tests mit Systemen mit mehreren Rechnern (v. a. Replikation, größere $N_{scenarios}$) durchgeführt werden. Dazu soll Cassandra v.5 (Dez. 2023) mit seinem überarbeiteten Speichermanagement verwendet werden.

In einem neuen Projekt sollen zudem entsprechende Datensätze und auch das Deployment offen gestellt werden im Sinne von Open Educational Resources.

Danksagung Die Autor:innen danken den anonymen Reviewer:innen für ihre wertvollen Hinweise. Diese Arbeit wurde gefördert durch das Bundesministerium für Bildung und Forschung in den Projekten „Wasserstoffleitprojekt TransHyDE_FP1 MechaMod“, Förderkennzeichen 03HY201N und „FHprofUnt FlexHyX“, Förderkennzeichen 13FH172PX8.

Funding Open Access funding enabled and organized by Projekt DEAL.

Interessenkonflikt T. Clees, M. Bareev-Rudy und M. Pfennig geben an, dass kein Interessenkonflikt besteht.

Open Access Dieser Artikel wird unter der Creative Commons Namensnennung 4.0 International Lizenz veröffentlicht, welche die Nutzung, Vervielfältigung, Bearbeitung, Verbreitung und Wiedergabe in jeglichem Medium und Format erlaubt, sofern Sie den/die ursprünglichen Autor(en) und die Quelle ord-

nungsgemäß nennen, einen Link zur Creative Commons Lizenz beifügen und angeben, ob Änderungen vorgenommen wurden.

Die in diesem Artikel enthaltenen Bilder und sonstiges Drittmaterial unterliegen ebenfalls der genannten Creative Commons Lizenz, sofern sich aus der Abbildungslegende nichts anderes ergibt. Sofern das betreffende Material nicht unter der genannten Creative Commons Lizenz steht und die betreffende Handlung nicht nach gesetzlichen Vorschriften erlaubt ist, ist für die oben aufgeführten Weiterverwendungen des Materials die Einwilligung des jeweiligen Rechteinhabers einzuholen.

Weitere Details zur Lizenz entnehmen Sie bitte der Lizenzinformation auf http://creativecommons.org/licenses/by/4.0/deed.de.

Literatur

Apache Software Foundation (2024) Apache Cassandra v.4.1, Documentation: "Cassandra / Data modelling / Evaluating and refining data models". https://cassandra.apache.org/doc/latest/cassandra/data_modeling/data_modeling_refining.html

Apache Software Foundation (2019) Issues Cassandra/CASSANDRA-8099—Refactor and modernize the storage engine. https://issues.apache.org/jira/browse/CASSANDRA-8099 (Erstellt: 6.2019)

Bareev-Rudy M (2020) Reliable NiFi Workflows for photovoltaic data. Hochschule Bonn-Rhein-Sieg, ". Masterprojektbericht (Enthalten in: FlexHyX (2024))

Bareev-Rudy M, Meiswinkel S, Pfennig M, Schedler S, Schiffer B, Steinebach G, Clees T (2023) Analysis of PtGtX systems with metal hydride storage based on coupled electrochemical and thermodynamic simulation. In: Procs. 18th conf. Sustainable development of energy, water and environment systems (SDEWES) Sep 24–29, 2023 (Revised and extended version submitted to Energy Conversion and Management)

Bray ET (2017) The JavaScript Object Notation (JSON) Data Interchange Format. Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc8259 (Erstellt: 12.2017)

Bray T et al (2008) Extensible Markup Language (XML) 1.0 (Fifth Edition). World Wide Web Consortium (W3C). https://www.w3.org/TR/xml/ (Erstellt: 02.2008)

Carpenter J, Hewitt E (2020) Cassandra, the definitive guide. O'Reilly Media

Cologne Intelligence (2019) Fast Data mit dem SMACK Stack. https://www.cologne-intelligence.de/blog/fast-data-mit-dem-smack-stack (Erstellt: 9. Mai 2019)

DataStax (2020a) CREATE KEYSPACE | CQL for Cassandra 3.0. https://docs.datastax.com/en/cql-oss/3.3/cql/cql_reference/cqlCreateKeyspace.html (Erstellt: 02.2020)

DataStax (2020b) CREATE TABLE | CQL for Cassandra 3.0. https://docs.datastax.com/en/cql-oss/3.3/cql/cql_reference/cqlCreateTable.html (Erstellt: 02.2020)

Estrada R (2016) Fast data processing systems with SMACK stack. Packt

FlexHyX (2024) Flexibilitätsoptionen regenerativer Wasserstofferzeugung und -nutzung mittels dezentraler stationärer Metallhydridspeicher und der Integration in Gasnetze. BMBF FHProfUnt-Projekt (Abschlussbericht)

Gomez JM et al (2018) Datenarchivierung von Energiedaten (DAvE). Projektdokumentation. https://uol.de/f/2/dept/informatik/download/lehre/PGs/PG-DAvE-neu.pdf?v=1528790921

Hagenmeyer V (2019) Energy Informatics in energy lab 2.0—A research platform for the energy transition. http://www.fze.uni-saarland.de/AKE_Archiv/AKE2019F/Vortraege/AKE2019F_03Hagenmeyer_EnergieInformatik_20190321.pdf

Heliocentris Academia International (2020) New Energy Lab V1.5, Experiment Guide, 2020. System an der Hochschule Bonn-Rhein-Sieg seit 2022 in Betrieb. https://www.heliocentrisacademia.com/product/new_energy_lab

ISO/IEC 20922 (2016) Information technology—Message Queuing Telemetry Transport (MQTT) v3.1.1", 2016-06-15. Edited by Andrew Banks und Rahul Gupta. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718009

James S (2019) Docker build: a beginner's guide to building docker images. https://stackify.com/docker-build-a-beginners-guide-to-building-docker-images/ (Erstellt: 07.2019)

Kerö N, Puhm A, Kernen T, Mroczkowski A (2019) Performance and reliability aspects of clock synchronization techniques for industrial automation. In: Procs. IEEE, S 1011–1026 https://doi.org/10.1109/JPROC.2019.2915972

Kinsta (2023) Was ist Docker? Ein vollständiger Leitfaden. https://kinsta.com/de/wissensdatenbank/was-ist-docker/

Koshy J (2016) Kafka Ecosystem at LinkedIn. https://engineering.linkedin.com/blog/2016/04/kafka-ecosystem-at-linkedin

Liu J (2023) Ein generisches und hoch skalierbares Framework zur Automatisierung und Ausführung wissenschaftlicher Datenverarbeitungs- und Simulationsworkflows. KIT-Fakultät für Informatik des Karlsruher Instituts für Technologie (KIT) (Dissertation)

Marz N, Warren J (2015) Big data: principles and best practices of scalable realtime data systems. Manning, New York

Monsberger (2020) User Feedback for Energy Efficiency in Buildings. Projekt FEELings. https://nachhaltigwirtschaften.at/resources/sdz_pdf/schriftenreihe-2020-39-feelings.pdf (Abschlussbericht)

Naik N (2017) Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. 2017 IEEE International Systems Engineering Symposium (ISSE, Vienna, S 1–7 https://doi.org/10.1109/SysEng.2017.8088251 (https://core.ac.uk/pdf/aaa160743474.pdf)

Pfennig M (2021) Performance Analyse der Datenübertragung zwischen MQTT und Kafka im Rahmen des Forschungsprojektes FlexHyX. Hochschule Bonn-Rhein-Sieg, Masterprojektbericht (Enthalten in: FlexHyX (2024))

Rinaldi S, Ferrari P, Flammini A, Sisinni E, Sauter T (2017) Network synchronization: an introduction". Wiley Encycl Electr Electron Eng 2017:1–12. https://doi.org/10.1002/047134608X.W8341

Shafranovich Y (2005) Common Format and MIME Type for Comma-Separated Values (CSV) Files. Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc4180 (Erstellt: 10.2005)

Wu H, Shang Z, Wolter K (2019) Performance Prediction for the Apache Kafka Messaging System, 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Zhangjiajie, China, 2019, pp. 154–161, https://doi.org/10.1109/HPCC/SmartCity/DSS.2019.00036

Hinweis des Verlags Der Verlag bleibt in Hinblick auf geografische Zuordnungen und Gebietsbezeichnungen in veröffentlichten Karten und Institutsadressen neutral.