Zentrum für sichere Informationstechnologie – Austria
Secure Information Technology Center – Austria

A-SIT

Seidlgasse 22 / 9, 1030 Wien
Tel.: (+43 1) 503 19 63–0
Fax: (+43 1) 503 19 63–66

Inffeldgasse 16a, 8010 Graz
Tel.: (+43 316) 873-5514
Fax: (+43 316) 873-5520

http://www.a-sit.at
E-Mail: office@a-sit.at

DVR: 1035461
ZVR: 948166612
UID: ATU60778947

VERTEILTES EVENTSYSTEM

Version 1.0 vom 07.01.2019
Bernd Prünster – bernd.pruenster@a-sit.at

Abstract/Zusammenfassung: Der Einsatz verteilter Systeme bedingt immer ein Mindestmaß an Parallelität. Einhergehend damit ergibt sich fast immer die Notwendigkeit, dass mehrere Instanzen auf einem gemeinsamen Datensatz arbeiten. Dies ist beispielsweise bei Cloudspeicherdiensten mit Offline-Caching und Synchronisationsmöglichkeiten der Fall. Im Rahmen dieses Projekts wurden Konzepte zur Umsetzung eines verteilten Eventsystems evaluiert und auf deren Basis ein Demonstrator entwickelt, welcher Publish-Subscribe-Elemente mit Ansätzen aus dem Blockchain-Umfeld vereint. Grundlegende Voraussetzungen an derartige Systeme sind die Schaffung einer gemeinsamen Netzwerkzeit, bzw. Ereignisreihenfolge. Besonderes Augenmerk wurde auf Replay-Möglichkeiten, Rollback und automatische Konfliktbehebung gelegt. Darüber hinaus wurde auch Performance-Aspekten und Netzwerklast besondere Aufmerksamkeit gewidmet, um ein möglichst effizientes System zu konzipieren. Des Weiteren wurde besonderer Wert darauf gelegt, dass die Umsetzung möglichst anwendungsunabhängig erfolgt.

Inhaltsverzeichnis

Inhaltsverzeichnis 1
1. Einleitung 2
2. Hintergrund 2
2.1. Lamport-Uhren und Happened-Before-Relationen 2
2.1.1. Eigenschaften der Happened-Before-Relation 3
2.2. Message Queues und Publish-Subscribe-Systeme 4
3. Verteiltes PubSub-basiertes Eventsystem 4
3.1. Grobarchitektur 5
3.2. Verteilter Broker, Ausfallsicherheit, und automatisierte Konfliktbehebung 5
3.3. Performance-Aspekte 7
3.4. Sicherheitsaspekte 8
4. Fazit 8
Referenzen 9

1. Einleitung

Der Einsatz verteilter Systeme geht immer mit Nebenläufigkeit und Asynchronität einher. Die Herausforderung liegt in diesem Zusammenhang in der Koordination der beteiligten Komponenten, insbesondere im Rahmen von Echtzeitanwendungen. Im Rahmen dieses Projekts wurden Konzepte für die verteilte Verarbeitung von Datensätzen betrachtet. Im Kern verfolgen solche Ansätze die Schaffung einer einheitlichen Ereignisreihenfolge über alle Teilnehmer hinweg (wobei Teilnehmer in diesem Kontext Instanzen der verteilten Anwendung beschreibt), sodass jederzeit Einigkeit über den aktuellen Zustand des Netzwerks herrscht und Inkonsistenzen vermieden werden. Ein Konzept, welches in der jüngeren Vergangenheit besondere Aufmerksamkeit auf sich gezogen hat, ist die ursprünglich im Rahmen der Kryptowährung *Bitcoin* [1] entwickelte *Blockchain*, welche als zentrales Transaktionsregister fungiert. Grund für das Interesse an der Technologie, sowie Kryptowährungen an sich, ist unter anderem, dass derartige Systeme kein Vertrauen zwischen den einzelnen Teilnehmern voraussetzen, ohne eine zentrale Instanz zu benötigen. Allerdings wird diese Eigenschaft zum Preis rechenintensiver Konsensmechanismen erreicht, und die Anwendungsmöglichkeiten abseits von Kryptowährungen sind stark eingeschränkt. Betrachtet man die Blockchain entkoppelt von Konsensverfahren, lässt sich das darunterliegende Konzept auf eine kryptografische Verkettung von zeitlich, bzw. logisch aufeinanderfolgenden Ereignissen reduzieren.

Ausgehend davon wird nachfolgend der technische Hintergrund, sowie Anforderungen verteilter Eventsysteme beleuchtet. Auf dieser Basis wurden im Rahmen dieses Projekts ein entsprechendes System konzeptioniert und ein Demonstrator entwickelt, welche in Abschnitt 3 beschrieben werden. Ziel war insbesondere auf zentrale Komponenten zu verzichten und ein vollkommen dezentral organisiertes Eventsystem zu entwickeln, welches als Basis für komplexere verteilte Anwendungen herangezogen werden kann. Abschnitt 4 fasst die Projektergebnisse zusammen und gibt einen Ausblick auf Weiterentwicklungsmöglichkeiten des umgesetzten Systems.

2. Hintergrund

Grundsätzlich kann in einem verteilten System der Aufbau widerspruchsfreier Kausalitätsketten als Voraussetzung für deterministisches Verhalten und Konsistenz angesehen werden. Dies kann durch den Einsatz so genannter *logischer Uhren* erreicht werden, welche Aktionen, bzw. Ereignisse in einem verteilten System relativ zueinander ordnen, und so eine *Happened-Before*-Relation abbilden. Exemplarisch wird nachfolgend das Prinzip der *Lamport-Uhr* erläutert, welche 1978 von Leslie Lamport definiert wurde [2] und die Grundlage für all diese Konzepte bildet. In diesem Zusammenhang wird auch der Aufbau einer Happened-Before-Relation im Rahmen von Blockchain-basierten Systemen diskutiert. Anschließend werden *Message Queues* und *Publish-Subscribe-Systeme* vorgestellt und im nachfolgenden Abschnitt 3 ein Konzept präsentiert, welches beide Ansätze in einem verteilten Eventsystem vereint.

2.1. Lamport-Uhren und Happened-Before-Relationen

Eine Lamport-Uhr ist ein Verfahren, eine logische Uhr in einem verteilten System zu betreiben. Konzeptionell handelt es sich dabei um einen Zähler, welcher von jedem Teilnehmer vor jeder die anderen Teilnehmer betreffenden Aktionen inkrementiert wird. Ausgehend von nachrichtenbasierter Kommunikation innerhalb des Systems muss jeder Teilnehmer beim Senden und Empfangen von Nachrichten diesen Zähler inkrementieren. Des Weiteren ist der aktuelle Zählerstand in jede ausgehende Nachricht einzubetten. Dadurch entfällt die Notwendigkeit zusätzlicher Synchronisationsnachrichten, sowie einer gemeinsamen Netzwerkzeit. Tatsächlich gibt es in einem solchen System keine Notwendigkeit für Zeitmessungen im eigentlichen Sinne, da üblicherweise lediglich eine Kausalordnung garantiert werden muss.
Beim Empfangen von Nachrichten wird der eigene Zähler mit dem Zeitstempel der empfangenen Nachricht verglichen. Anschließend wir der eigene Zähler auf den größeren der beiden Werte gesetzt und inkrementiert. Daraus lässt sich folgendes ableiten: Wenn ein Ereignis (bzw. eine Nachricht) *a* ein Ereignis *b* beeinflussen kann, muss der Zähler *C(a)* kleiner als der Zähler *C(b)* sein. Dies wir als *schwaches Konsistenzkriterium für Uhren*, bzw. als *Happened-Before-Relation*

bezeichnet. Listing 1 und Listing 2 fassen die Sende- und Empfangsprozeduren in Form von Pseudo-code zusammen.

```
counter +=1
send(message, counter)
```

Listing 1: Prozedur für das Senden von Nachrichten bei Verwendung einer Lamport-Uhr

```
(message, counter_rxd) = receive()
counter = max(counter, counter_rxd) +1
```

Listing 2: Prozedur für das Empfangen von Nachrichten bei Verwendung einer Lamport-Uhr

Im Kontext verteilter Systeme, welche Lamport-Uhren oder davon abgeleitete Verfahren verwenden, lassen sich die nachfolgend beschriebenen Eigenschaften ableiten.

2.1.1. Eigenschaften der Happened-Before-Relation

Da es sich bei der Happened-Before-Relation (Notation: →) mathematisch um eine strikte Halbordnung handelt, ist diese transitiv, irreflexiv, und antisymmetrisch, d.h. für Ereignisse, bzw. Nachrichten a, b, c gilt:

$\forall a, b, c: (a \rightarrow b) \wedge (b \rightarrow c) \Rightarrow a \rightarrow c$	Ist *a* vor *b* passiert, und *b* vor *c*, ist auch *a* vor *c* passiert
$\forall a: a \nrightarrow a$	Ein Ereignis kann nicht vor sich selbst stattgefunden haben
$\forall a, b: (a \neq b) \wedge (a \rightarrow b) \Rightarrow b \nrightarrow a$	Ist *a* vor *b* passiert, kann *b* nicht vor *a* passiert sein

Im Kontext verteilter Systeme ergeben sich aus diesen Definitionen entscheidende Konsequenzen: Aus *a* →*c* und *b* →*c* lässt sich nichts über die semantischen Zusammenhänge zwischen Ereignissen ableiten und somit nicht klären, ob es sich dabei auch tatsächlich um einen Konflikt handelt (oder es keinen kausalen Zusammenhang zwischen den Nachrichten gibt). Eine weitere Facette dieser Eigenschaft ist, dass es keine Möglichkeit gibt, Nebenläufigkeit zu modellieren, da alle Ereignisse als lineare Folge abgebildet werden.
Obwohl es sich hierbei eindeutig um Limitierungen handelt, auf Grund derer Lamport-Uhren für eine Reihe von Anwendungen ausscheiden, reicht es oftmals aus, wenn ein verteiltes System lediglich die schwache Konsistenzbedingung für Uhren erfüllt. Als bekannte Beispiele für Systeme, welche auf einer lineare Ereignisfolge basieren, sind in diesem Zusammenhang das zentrale Versions-kontrollsystem *Apache Subversion* [3], sowie moderne Kryptowährungen auf Blockchain-Basis zu nennen.

Blockchain-basierte Systeme stützen sich im Kern auf eine *append-only* Datenstruktur. Die Reihenfolge von Ereignissen, bzw. Blöcken erfolgt durch kryptografische Verkettung: Jeder Block, bzw. jedes Ereignis referenziert anstatt eines Zählers den Vorgängerblock, bzw. das vorhergehende Ereignis über dessen kryptografischen Hash, woraus sich ebenfalls eine Happened-Before-Relation ergibt. Im Gegensatz zu logischen Uhren, ist es jedoch nicht ohne weitere Informationen möglich, zwei nicht direkt verkettete Blöcke zueinander in Relation zu setzen, da ausgehend vom Verweis auf den vorangegangenen Block an Hand eines Hashwerts, ohne Kenntnis dieses Vorgängerblocks keine Reihung von Blöcken möglich ist. Folglich kann ein Teilnehmer eines Blockchain-basierten Systems ohne Kenntnis der vollständigen Kette von bisher veröffentlichten Blöcken neu eingehende Blöcke nicht ohne Weiteres einordnen.
Bei der Blockchain handelt es sich zudem um eine Sonderform von gerichteten, azyklischen Merkle-Graphen, welche beispielsweise auch in verteilten Versionsverwaltungssystemen wie *Git* [4] zum Einsatz kommen. Durch die Erweiterung auf einen gerichteten, azyklischen Graphen lässt sich (im Gegensatz zu einer linearen Kette) Nebenläufigkeit modellieren, wobei Wissen um die Semantik von Ereignissen entscheidend ist, um Konflikte lösen zu können. Eine Möglichkeit, derartige Probleme einzugrenzen, besteht darin, im Rahmen jeder innerhalb eines verteilten System ausgetauschten Nachricht zu deklarieren, welche Daten ein Ereignis betrifft, sodass zwar keine zweifelsfreien

Kausalitätsketten gebildet werden können, jedoch zumindest die Unabhängigkeit von Ereignissen, welche unterschiedliche Daten betreffen, eindeutig feststellbar ist.

Der datenspezifische bzw. themenbezogene Austausch von Daten kann im Rahmen verteilter Systeme als so genanntes *Publish-Subscribe-Pattern* modelliert werden. Der nachfolgende Abschnitt beschreibt dessen Funktionsweise. Darauf und auf den im Rahmen dieses Abschnitts diskutierten Konzepten wird in Abschnitt 3 ein dezentrales verteiltes Eventsystem vorgestellt, welches als Basis komplexerer verteilter Systeme eingesetzt werden kann.

2.2. Message Queues und Publish-Subscribe-Systeme

Im Kontext verteilter Systeme beschreiben *Message Queues* einen (typischerweise) asynchronen Kommunikationsmechanismus; einen gemeinsamen Nachrichtenbus, der von allen Teilnehmern zum Austausch von Nachrichten untereinander verwendet wird. Im Wesentlichen werden Nachrichten im Bus zwischengespeichert, bis sie vom Empfänger (ähnlich einer Mailbox) abgerufen werden. Dadurch ist keine direkte Interaktion zwischen Sender und Empfänger notwendig.

Eine Erweiterung dieses Ansatzes stellt das Publish-Subscribe-Modell („PubSub“) dar, welches Nachrichten nicht in einer gemeinsamen Warteschlange reiht, sondern anhand ihrer Eigenschaften gruppiert und an Interessenten verteilt. Wie Message Queues entkoppeln auch PubSub-Systeme Sender (welche als *Publisher* bezeichnet werden) von Empfängern (welche entsprechend *Subscriber* genannt werden).
Im einfachsten Fall kommt eine zentrale Komponente, der so genannte *Broker* zum Einsatz, welcher für die Verteilung von Nachrichten an diejenigen Empfänger zuständig ist, welche Interesse an Nachrichten mit bestimmten Eigenschaften bekundet haben. Die Verantwortlichkeit der Kategorisierung kann (je nach Art des Pub-Sub-Systems) entweder von Publishern, oder Subscribern erfolgen. Nachfolgend wird exemplarisch die Kategorisierung von Nachrichten an Hand von vom Publisher definierten Kriterien illustriert.

Im Rahmen so genannter *kriterien-*, bzw. *themenbasierter* PubSub-Systeme deklarieren Publisher für jede versendete Nachricht eine Reihe von Kriterien, bzw. Themen auf die sich die jeweilige Nachricht bezieht und übermittelt diese anschließend an den Broker. Subscriber registrieren sich am Broker, um Benachrichtigungen über neue Nachrichten betreffend bestimmter Kriterien zu erhalten. Sobald Nachrichten am Broker eintreffen, werden diese an all jene Subscriber weitergeleitet, welche Interesse an den (von Publishern definierten) Kriterien der Nachricht bekundet haben.
Durch den Einsatz eines Brokers ergibt sich eine klassische Stern-Topologie, bzw. ein Client-Server-Modell mit allen Vor- und Nachteilen. In diesem Zusammenhang ist anzumerken, dass weder Message Queues, noch PubSub-Systeme als inhärent Peer-to-Peer oder dezentral konzeptioniert sind. Folglich ist die Abhängigkeit von einer zentralen Instanz nicht per se als nachteilig anzusehen, sondern vielmehr Teil des Konzepts. Dasselbe gilt auch für verteilte Systeme im Allgemeinen, welche oftmals zentral verwaltet werden. Unter Anderem im PubSub-Fall entfällt zumindest konzeptionell der Bedarf nach Synchronisation, da Sortierung und Konfliktbehebung vom Broker durchgeführt werden können. Inwieweit eine solche Strategie sinnvoll ist, hängt im Detail jedoch von der konkreten Anwendung ab.

Nachfolgend wird das im Rahmen dieses Projekt entwickelte verteilte Eventsystem beschrieben, welches kryptografische Verkettung von Ereignissen mit PubSub-Konzepten verbindet. Dabei handelt es sich im ein bewusst anwendungsunabhängig gestaltetes Framework, das auf einem strukturierten Peer-to-Peer-Netz basiert.

3. Verteiltes PubSub-basiertes Eventsystem

Im Rahmen dieses Projekts wurde ein verteiltes Eventsystem entwickelt, welches keine zentrale Instanz (wie z.B. einen Broker) erfordert. Die Grundlage bildet ein strukturiertes Peer-to-Peer-Netzwerk auf Kademlia- [5] bzw. S/Kademlia-Basis [6]. Prinzipiell kann jedoch jedes beliebige strukturierte Peer-to-Peer-Netz als Unterbau eingesetzt werden. Die Grundkonzepte des

entwickelten Systems sind darüber hinaus bewusst schlicht und anwendungsunabhängig gehalten, sodass diese einerseits auf bestehende Peer-to-Peer-Netze angewandt werden können, andererseits auch beliebige verteile Anwendungen realisiert werden können, welche eine dezentral organisierte, gemeinsame Datenbasis erfordern. Besonderes Augenmerk wurde auf Effizienz und Nebenläufigkeit gelegt. Um dieses Potential voll auszuschöpfen, wurde ein eigens dahingehend optimiertes Peer-to-Peer-Netz implementiert, sodass moderne, hochgradig effiziente Concurrency-Features der verwendeten Programmiersprache (konkret: *Kotlin-Coroutinen* [7]) auf allen Ebenen der Implementierung eingesetzt werden können. Dadurch können die verfügbaren System- und Netzwerkressourcen maximal ausgenutzt werden, ohne dass dies die Lesbarkeit und Integrierbarkeit der Codebasis negativ beeinflusst.

3.1. Grobarchitektur

Konzeptionell wird als Basis für das entwickelte verteilte Eventsystem ein sicheres[1], strukturiertes Peer-to-Peer-Netzwerk mit grundlegender Replikationsfunktionalität angenommen. Konkret ist eine Entkopplung der physischen Netzwerkstruktur und die Verwendung logischer Identifikatoren für Netzwerteilnehmer und Daten, anstatt IP-basierten Routings notwendig. Zusätzlich muss das verwendete Peer-to-Peer-Netzwerk das Auffinden mehrerer Netzwerkknoten, die einem Identifikator am nächsten, sind unterstützen und insbesondere nicht nur *den dem Identifikator am nächsten liegenden* Knoten produzieren und für die Weiterverarbeitung am Application-Layer zugänglich machen.
Ausgehend davon werden Nachrichten, bzw. Events von Ersteller einem Kriterium zugeordnet, welches Teil desselben Identifikatorraums wie die Netzwerkteilnehmer sind. Um dies zu erreichen, können Kriterien zwar vollkommen frei gewählt werden, abgebildet werden diese jedoch jeweils als deren kryptografischer Hashwert. Nachrichten werden anschließend an diejenigen Netzwerkteilnehmer gesendet, die einem Kriterium-Hash am nächsten sind. Die Empfänger speichern eigehende Events und stellen diese dem restlichen Netzwerk auf Abruf bereit.
Einem Kriterium zugehörige Nachrichten sind untereinander kryptografisch verkettet, wodurch eine eindeutige Ereignisreihenfolge abgeleitet werden kann. In diesem Aspekt handelt es sich im dieselbe Vorgehensweise wie sie im Rahmen von Kryptowährungen Anwendung findet, ohne jedoch aufwändige Konsensverfahren zu erzwingen. Rollback-Funktionalität ergibt sich wie bei allen append-only Datenstrukturen per Definition auf Grund der Datenstruktur und bedarf keiner weiteren Implementierung. Nebenläufigkeit ist durch diese Konstruktion auf der Granularitätsebene von Kriterien möglich, da es keine Verkettung, bzw. gemeinsame Sortierung von Ereignissen betreffend unterschiedlicher Kriterien gibt.
Haben andere Knoten Interesse an Informationen zu einem bestimmten Kriterium, können diese entsprechende Anfragen an die für die Speicherung der betreffenden Ereignisse zuständigen Knoten absetzen. Dadurch und auf Grund der replizierten Speicherung von Ereignissen können Offline-Phasen überbrückt und Replay-Maßnahmen bereitgestellt werden (siehe Abschnitt 3.2).

Ein Subscribe-Mechanismus wie in vollwertigen PubSub-Systemen ist aus Performance-Gründen nicht als zwingend erforderlich konzipiert und im entwickelten Demonstrator auch nicht implementiert (siehe Abschnitt 3.3). Das verteilte Speichern von Ereignissen entspricht jedoch ungeachtet dessen konzeptionell dem Einsatz eines verteilten Brokers. Die dahinterstehenden Überlegungen und die daraus resultierenden Eigenschaften werden im nachfolgenden Abschnitt diskutiert.

3.2. Verteilter Broker, Ausfallsicherheit, und automatisierte Konfliktbehebung

Die verteilte Broker-Komponente des entwickelten Systems ist vollständig dezentral organisiert, bzw. ergibt sich direkt aus den versendeten Nachrichten. Dadurch, dass Kriterien auf Hashwerte abgebildet werden, ist nicht vorhersehbar, welche Teilnehmer als Broker für bestimmte Nachrichten fungieren werden. In jedem Fall ergibt sich jedoch eine diesbezügliche Gleichverteilung im Netzwerk

[1] Als *sicher* wird in diesem Kontext eine gewisse Mindestresistenz gegenüber Sybil- [11] und Eclipse-Attacken [8] angesehen. Details können einem vorangegangenen A-SIT-Projekt zum Thema sichere Peer-to-Peer-Netze [9] und einer in diesem Rahmen entstandenen wissenschaftlichen Publikation [10] entnommen werden.

(auf Grund der Eigenschaften kryptografischer Hashfunktionen, gleichverteilte Werte zu produzieren). Ab dem Zeitpunkt, ab dem ein Netzwerkknoten für ein Kriterium zuständig ist, kann dieser jedoch herausfinden, welche Knoten außerdem noch für dieses Kriterium verantwortlich sind. In Folge dessen können sich zuständige Knoten untereinander synchronisieren.
Verlässt ein Knoten das Netzwerk, muss dies nicht explizit kommuniziert werden: Sobald neue Ereignisse bezüglich eines Kriteriums produziert werden, übernimmt automatisch ein anderer Knoten die Funktion von zuvor weggefallenen Netzwerkteilnehmern. Dies ergibt sich aus der Funktionsweise des darunterliegenden Peer-to-Peer-Netzwerks:

1. Für jedes zu speichernde Ereignis werden immer gleich viele dem Kriterium-Hash am nächsten liegende Knoten für dessen Speicherung ausgewählt.
2. Empfängt ein Knoten ein Ereignis zur Speicherung, dessen Kriterium ihm entweder nicht bekannt ist, oder dessen Vorgänger lokal nicht gespeichert ist, wird eine Synchronisationsanfrage an alle anderen dem Kriterium am nächsten liegenden Teilnehmer gesendet.
3. Der Knoten kann aus den Antworten auf die Synchronisationsanfrage die vollständige Ereigniskette rekonstruieren und unterscheidet sich anschließend nicht mehr von den anderen, bereits länger im Netzwerk vorhandenen und für ein Kriterium zuständigen, Netzwerkknoten.

Diese Strategie wurde gewählt, da sie kaum Funktionalität beinhaltet, welche nicht ohnehin von de facto jedem strukturierten Peer-to-Peer-Netzwerk unterstützt wird. Tatsächlich sind durch die Zuweisung der Broker-Rolle auf Knoten an Hand ihrer Identifikatoren in Kombination mit der verwendeten append-only Datenstruktur sowohl Ausfallsicherheit, als auch Replay-Funktionalität gegeben. Welches Maß an Ausfallsicherheit gewünscht ist, lässt sich direkt skalieren, indem mehr oder weniger Knoten als Broker für jeweils ein Kriterium herangezogen werden. Dies wird über den in diesem Zusammenhang als *Replikationsfaktor* bezeichneten Parameter festgelegt. Hierbei handelt es sich sinnvollerweise um einen globalen Parameter, der im gesamten Netzwerk denselben Wert haben muss, da zuvor beschriebenes Prozedere sonst zu Inkonsistenzen führen kann.

Ein weiterer Grund, weshalb der Replikationsfaktor auf einen global eindeutigen Wert festgelegt werden muss, ergibt sich aus der gewählten Konfliktlösungsstrategie. Diese basiert im Wesentlichen auf einem Mehrheitsentscheid im Rahmen des Synchronisationsprozesses, welcher wiederum nur bei Bedarf als Teil der Publish-Operation angestoßen wird. Diese ist wie folgt definiert:

1. Jedem Teilnehmer steht es frei, jederzeit zu jedem beliebig wählbaren Kriterium Ereignisse an die dafür zuständigen Netzwerkknoten zu senden.

2. Diese Operation ist erfolgreich wenn:

    a. es sich um ein Kriterium handelt, zu dem bisher keine Information vorhanden ist, oder
    b. der Publisher im produzierten Ereignis das vorhergehende Ereignis an Hand dessen Hashwert referenziert.

3. Für den Fall, dass einzelnen Broker-Knoten das vorhergehende Ereignis nicht bekannt ist, stellen diese eine Anfrage an alle anderen für das Kriterium zuständigen Broker nach allen Ereignissen, die neuer als das letzte ihnen bekannte ist. Diese Synchronisation erfolgt jedenfalls bevor eine Antwort (Bestätigung, oder Fehler) auf die ursprüngliche Publish-Operation retourniert wird.

4. In jedem Fall antworten Broker-Knoten unabhängig voneinander entweder mit einer Bestätigung, oder mit einer Fehlermeldung:

    a. Handelt es sich bei der Mehrheit der Antworten um Bestätigungen, wird die Publish-Operation als erfolgreich angesehen.
    b. Antwortet die Mehrheit der Broker mit einer Fehlermeldung, welche besagt, dass es sich bei dem als vorhergehendes Ereignis referenzierten Ereignis, nicht um das

aktuell Letztbekannte handelt, geht der Publisher wie folgt vor:

i. Der Publisher sendet eine Synchronisationsanfrage analog zu 3. an alle Broker-Knoten.
ii. Nach der Synchronisation ist zu überprüfen, ob die aktualisierte Datenbasis die ursprünglich geplante Aktion nach wie vor zulässt, oder ob diese zu verwerfen ist.
iii. Schließlich sendet der Publisher ein neues Ereignis an die Broker, welches zum aktuellen Datenbestand an den Broker-Knoten passen sollte.
iv. Haben zwischenzeitlich andere Netzwerkteilnehmer neue Ereignisse zum betreffenden Kriterium produziert, sind die Schritte i bis iv zu wiederholen.

Automatische Update-Benachrichtigungen betreffend Kriterien (eine vollwertige Subscribe-Funktionalität) sind nicht als Kernfunktionalität vorgesehen (können jedoch nachträglich ohne großen Aufwand hinzugefügt werden). Diese Einschränkung hat keinen Einfluss auf die Konzepte des umgesetzten dezentralen Eventsystems. Vielmehr handelt es sich dabei um eine bewusste Entscheidung, welche der Systemperformance zugutekommt und unnötige Netzwerklast verhindert.

3.3. Performance-Aspekte

Die Entscheidung gegen ein vollwertiges PubSub-System inklusive Subscribe-Funktionalität basiert im Kern auf folgender Beobachtung: Es gibt eine Klasse verteilter Anwendungen, welche lediglich dann eine aktuelle Datenbasis betreffend bestimmter Kriterien benötigen, wenn neue Informationen betreffend diese Kriterien produziert werden soll. Eine alternative Interpretation ist folgende: Es gibt Klassen von Anwendungen, bei denen nur im Fall eines Schreibzugriffs eine aktuelle Datenbasis vonnöten ist. Unabhängig davon kann es immer die Notwendigkeit von gezielten, und/oder periodischen Synchronisationsanfragen (Polling) geben.

Ausgehend davon ergibt sich ein potentiell drastischer Performance-Gewinn, wenn man auf vollwertige Subscribe-Funktionalität verzichtet. Egal wie sehr sich die Datenbasis zwischen Synchronisationsanfragen ändert, bzw. unabhängig davon, wie viele Ereignisse zwischen zwei Publish-Operationen eines einzelnen Knoten an den Broker-Knoten eingehen, müssen lediglich zwei (bzw. vier) Nachrichten zwischen Broker und diesem Knoten ausgetauscht werden, um eine aktuelle, konsistente Datenbasis zu erhalten. Alle in der Zwischenzeit an den Brokern eingegangenen Ereignisse können gesammelt im Rahmen einer einzigen Synchronisationsnachricht übertragen werden. Würde unter diesen Umständen ein vom Broker aktiv betriebener Subscribe-Mechanismus zum Einsatz kommen, ginge dies mit erhöhter Netzwerklast einher, deren Nutzen insofern begrenzt wäre, als das Subscriber Updates zum Kriterien erhielten, auch wenn dies nicht in Echtzeit notwendig wäre.

Unabhängig von der konzeptionellen Architektur wurde aus Performance-Gründen auch eine enge Kopplung an die dem Eventsystem untergeordnete Peer-to-Peer-Netzwerkimplementierung gewählt. Diese Entscheidung begründet sich mit der inhärenten Nebenläufigkeit und Asynchronität verteilter Systeme. Da dies sowohl das Peer-to-Peer-Netz als auch das darauf aufbauende Eventsystem betrifft, ergibt sich ein insgesamt effizienteres Gesamtsystem, wenn auf allen Ebenen dieselben Concurrency-Mechanismen zum Einsatz kommen, und die aktive (Client) als auch die reaktive (Server) Komponente der Peer-to-Peer-Netz-Implementierung direkt für den Austausch von Nachrichten des Eventsystems verwendet werden. Dieser Effekt wird zusätzlich durch den Einsatz moderner Concurrency-Mechanismen (Kotlin-Coroutinen) verstärkt, wodurch – verglichen mit traditioneller Thread-basierter Programmierung – (auch durch Compileroptimierungen) ein Leistungsgewinn um mehr als das Zehnfache erreicht werden konnte, bei gleichzeitig leichterer Lesbarkeit und Integrierbarkeit des Quellcodes.

Die enge Kopplung des Eventsystems an das darunterliegende Peer-to-Peer-Netz führt darüber hinaus zu einer strukturierteren Sicherheitsarchitektur des Gesamtsystems, sofern das Peer-to-Peer-Netzwerk als sicher (siehe Abschnitt 3.1) angenommen werden kann. Details hierzu werden nachfolgend diskutiert.

3.4. Sicherheitsaspekte

Die bisher beschriebene Funktionalität basiert auf der Annahme, dass sich die Mehrheit aller Netzwerkteilnehmer ehrlich verhält. Da sich dies auf technischer Ebene auf effektive Maßnahmen gegen Sybil- und Eclipse-Attacken reduzieren lässt (und operative und soziale Aspekte von Angriffen auf technische Systeme im Kontext dieses Systems nicht weiter betrachtet werden), kann ausgehend davon ein Großteil des Gesamtsicherheitsaspekts auf die Peer-to-Peer-Netzwerkebene delegiert werden.
Konkret baut der im Rahmen dieses Projekts entwickelte Demonstrator auf einer Peer-to-Peer-Implementierung auf, welche unter anderem selbstzertifizierende Identifikatoren verwendet (siehe vorangegangenes A-SIT-Projekt zum Thema [8]). Entsprechend wurde Accountability auf Eventsystem-Ebene vollständig ausgeklammert, da diese Funktionalität vom Peer-to-Peer-Netzwerk bereitgestellt wird. In Folge dessen lässt sich das hier beschriebene Konzept eines verteilten Eventsystems auch ohne großen Aufwand direkt auf populäre Peer-to-Peer-Netzwerke wie beispielsweise IPFS[2] übertragen, da lediglich allgemeine Konzepte als Voraussetzung angesehen werden, nicht jedoch konkrete Implementierungen davon. Zusätzlich kommt dieser Ansatz der Effizienz des Gesamtsystems insofern zu Gute, als dass bestehende Funktionalität herangezogen wird, und diese nicht erneut auf Eventsystemebene implementiert werden muss.

Ein weiterer Sicherheitsgewinn ergibt sich durch die Abbildung von Kriterien auf kryptografische Hashwerte: Ausgehend von Hashwerden kann nicht auf das Kriterium geschlossen werden, was Privatsphäreaspekten zugutekommt.

4. Fazit

Im Rahmen dieses Projekts wurden Konzepte zur Umsetzung eines dezentral organisierten, verteilten Eventsystems evaluiert. Ausgehen von append-only Datenstrukturen, wie sie im Rahmen moderne Kryptowährungen oder dem verteilten Versionskontrollsystem Git eingesetzt werden, und den Grundideen von Publish-Subscribe-Systemen, wurde ein Konzept entwickelt, welches beide Ansätze miteinander vereint. Ein entsprechender Demonstrator, welcher dieses Konzept umsetzt, veranschaulicht dessen Praxistauglichkeit.
Ereignisse werden (analog zu Nachrichten im Rahmen traditioneller PubSub-Systeme) an Hand von Kriterien kategorisiert und für jeweils jedes Kriterium (ähnlich einer Blockchain) kryptografisch miteinander verkettet. Dadurch lässt sich Nebenläufigkeit bezüglich Ereignissen unterschiedlicher Kriterien modellieren, Replay-, Rollback- und Replikationsfunktionalität bereitstellen, sowie durch den Einsatz eines sicheren Peer-to-Peer-Netzwerks als Basis des Eventsystems auch Konfliktbehebung ohne komplexe Konsensverfahren umsetzen. Generell wurde Anwendungsunabhängigkeit, sowie größtmögliche Effizienz bei gleichzeitiger Lesbarkeit des Quellcodes angestrebt. Durch den Einsatz einer Plattformunabhängigen Programmiersprache, moderner Concurrency-Mechanismen, sowie der Implementierung offener Schnittstellen konnten diese Ziele erreicht werden.
Zwar wurde keine vollwertige Subscribe-Funktionalität im Sinne eines PubSub-Systems umgesetzt, allerdings hat dies keinen Einfluss auf die Bewertung des Konzepts, da die Verwaltung von Subscribern analog zur in Abschnitt 3 beschriebenen Verwaltung und Synchronisation von Ereignissen umgesetzt werden kann. Entsprechend ist dies ein möglicher Ansatzpunkt für zukünftige Weiterentwicklungen des umgesetzten Demonstrators.

[2] https://ipfs.io/

Referenzen

[1] S. Nakamoto, „Bitcoin: A Peer-to-Peer Electronic Cash System,“ 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf. [Zugriff am 31 10 2018].

[2] L. Lamport, „Time, Clocks, and the Ordering of Events in a Distributed System,“ *Communications of the ACM,* Bd. 21, Nr. 7, pp. 558-565, 1978.

[3] CollabNet, Apache Software Foundation, „Apache Subversion,“ [Online]. Available: https://subversion.apache.org/. [Zugriff am 22 12 2018].

[4] L. Torvalds, J. Hamano und e. al., „Git,“ [Online]. Available: https://git-scm.com/. [Zugriff am 20 12 2018].

[5] P. Maymounkov und D. Mazières, „Kademlia: A Peer-to-Peer Information System Based on the XOR Metric,“ in *Peer-to-Peer Systems*, Springer, 2002, pp. 53-65.

[6] I. Baumgart und S. Mies, „S/Kademlia: A practicable approach towards secure key-based routing,“ in *2007 International Conference on Parallel and Distributed Systems*, 2007.

[7] JetBrains, „Coroutines for asynchronous programming and more,“ [Online]. Available: https://kotlinlang.org/docs/reference/coroutines-overview.html. [Zugriff am 22 12 2018].

[8] B. Prünster, „Sichere Peer-to-Peer-Netze auf Basis von Selbstzertifizierung,“ 07 05 2018. [Online]. Available: https://technology.a-sit.at/sichere-peer-to-peer-netze-auf-basis-von-selbstzertifizierung/. [Zugriff am 11 07 2018].

[9] R. Fantacci et al., „Avoiding Eclipse Attacks on Kad/Kademlia: An Identity Based Approach,“ in *2009 IEEE International Conference on Communications*, IEEE, 2009.

[10] B. Prünster, D. Ziegler, C. Kollmann und B. Suzic, „A Holistic Approach Towards Peer-to-Peer Security and why Proof of Work Won't Do,“ in *14th EAI International Conference on Security and Privacy in Communication Networks*, Singapur, Springer, 2018.

[11] J. R. Douceur, „The Sybil Attack,“ in *Peer-to-Peer Systems*, Berlin, Springer, 2002, pp. 251-260.