Institut für Parallele und Verteilte Systeme
Abteilung Verteilte Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3499

Zuverlässigkeit parallelisierter und verteilter CEP-Systeme

Julian Trischler

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Betreuer:	Dipl.-Inf. Ruben Mayer
begonnen am:	2013-06-01
beendet am:	2013-12-01
CR-Nummer:	C.2.4, C.4

Kurzfassung

Mithilfe des Paradigmas „Complex-Event-Processing“ (CEP) lassen sich auf großen Datenströmen aus einfachen Ereignissen durch Korrelationen höherwertige Informationen in Echtzeit ableiten. An CEP-Systeme werden hohe Anforderungen wie Skalierbarkeit, vollständig korrekte Ergebnisse auch im Fehlerfall sowie eine hohe Verfügbarkeit und rechtzeitige Situationserkennung gestellt. Für viele Anwendungsfälle eignen sich verteilte und parallelisierte CEP-Systeme. Bei der selektionsbasierten Operatorparallelisierung werden eintreffende Datenströme anhand von Domänenwissen so in Selektionen zum parallelen Weiterverarbeiten aufgeteilt, dass zu erkennende Muster immer in mindestens einer Selektion komplett vorliegen. In dieser Diplomarbeit wird die Zuverlässigkeit dieser Art der Parallelisierung unter den Gesichtspunkten der Rechtzeitigkeit und vollständigen Korrektheit untersucht. Dabei kommen zwei grundverschiedene Verfahren zum Einsatz: Replikation und Wiederherstellung von Selektionen im Fehlerfall. Auch bei Ausfall einer Operatorinstanz müssen Situationen korrekt und rechtzeitig erkannt werden. Schließlich wird der mit einem mathematischen Modell berechnete, möglichst optimale Parallelisierungsgrad experimentell überprüft und ausgewertet. Es stellt sich heraus, dass das Wiederherstellungsverfahren im Vergleich zur Replikation wirtschaftlicher ist und dabei alle abgegebenen Garantien einhalten kann.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

In vielen Bereichen des heutigen Lebens fallen große Datenmengen an, die möglichst in Echtzeit verarbeitet werden sollen. Aus vielen kleinen Informationsstücken soll eventuell über mehrere Schritte hinweg eine komplexere Situation erkannt werden. Zum Beispiel messen verschiedene Sensoren Temperatur, Luftfeuchtigkeit, Druck und andere Gaskonzentrationen in einer Chemiefabrik. Diese unterschiedlichen Messwerte werden an einen Kontrollrechner übermittelt, der darin verschiedene Muster sucht, um Gefahren zu erkennen. Wurde etwa eine außergewöhnliche Zusammensetzung der Luft entdeckt, kann Alarm geschlagen werden und beispielsweise die Werksfeuerwehr ausrücken. Dieser hier beschriebene Kontrollrechner kann durch ein sogannantes *Complex-Event-Processing-System* (CEP-System) realisiert sein.

Weitere Anwendungsmöglichkeiten solcher CEP-Systeme bieten sich in der Transportlogistik zur Positionsüberwachung von Containern [May12] oder in der Energiewirtschaft im Sinne von intelligenten Stromzählern [SKPR10]. Neben vielen weiteren Feldern eignen sie sich auch zur Steuerung von Fertigungsprozessen [SPR09] oder zur Börsenkurserkennung in der Finanzwirtschaft [BDWT13].

Entsprechend der vielfältigen Anwendungsbereiche werden an CEP-Systeme eine Reihe an Anforderungen gestellt. So müssen die produzierten Ergebnisse natürlich korrekt sein und möglichst schnell beim Interessenten eingehen. Zudem soll das System gegen Ausfälle robust sein. Auch die Skalierbarkeit spielt eine große Rolle. Sind Sensoren räumlich verteilt, ist ein verteiltes CEP-System eine gute Wahl, denn so können Ereignisse schon nah den Sensoren verarbeitet werden. Bei großen Datenmengen oder komplizierten Verarbeitungsschritten ist der Durchsatz zu niedrig, um die eingehenden Daten rechtzeitig zu verarbeiten; dies ist durch Parallelisierung in den Griff zu bekommen. Doch in parallelisierten und verteilten CEP-Systemen ist es schwieriger, korrekte Ergebnisse zeitnah und zuverlässig an den Interessenten auszuliefern. Die gesteigerte Komplexität macht es zudem auch nicht einfacher, das System ständig verfügbar zu halten.

Für die Parallelisierung der Verarbeitung gibt es verschiedene Ansätze. In dieser Arbeit soll der Selektionsbasierte zum Einsatz kommen. Hierbei werden die in den Verarbeitungsknoten eintreffenden Datenströme anhand eines vorhandenen Domänenwissens in sogenannte Selektionen aufgeteilt, sodass die zu erkennenden Muster immer in mindestens einem der Teilströme komplett enthalten sind [May13b, May13a].

Bisher gibt es zwei etablierte Grundverfahren, um beim Ausfall eines Rechenknotens sicher in den Normalbetrieb wieder zurückzukehren und am Ende keine Daten zu verlieren oder aufgrund eines Fehlers Zusätzliche zu erzeugen oder Bestehende zu modifizieren. Bei der Replikation werden alle zu bearbeitenden Daten mehrfach an verschiedene Rechenknoten zur Verarbeitung übergeben. Fällt eine Verarbeitungseinheit aus, liegen die Informationen immer

noch auf einer anderen vor [VKR11]. Der andere Ansatz ist, mithilfe von Sicherungspunkten verlorene Daten bei Bedarf wiederherzustellen [KMR+13].

Diese Arbeit untersucht die Zuverlässigkeit von parallelisierten und verteilten CEP-Systemen, bei denen selektionsbasiert parallelisiert wird. Insbesondere sollen Situationen korrekt und rechtzeitig erkannt werden, um so auf diese reagieren zu können. Es entsteht ein Rahmenwerk, das die Aufteilung in Selektionen für die parallele Verarbeitung erstmals ermöglicht. Konkret untersucht diese Diplomarbeit die Auswirkung des Parallelisierungsgrads auf die Warteschlangen der Verarbeitungseinheiten in Bezug auf Rechtzeitigkeit, wobei die beiden oben genannten Verfahren für die Wiederherstellung eingesetzt und miteinander verglichen werden. In verschiedenen Experimenten werden die anhand eines mathematischen Modells berechneten Lösungen überprüft und ausgewertet.

Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Einführung in das Complex-Event-Processing: Hier werden werden die allgemeinen Grundlagen dieser Diplomarbeit beschrieben. Es wird erklärt, was für Arten von CEP-Systemen es gibt und worin sie sich unterscheiden.

Kapitel 3 – Systemmodell zu verteilten und parallelisierten CEP-Systemen: Dieses Kapitel stellt ein formales Systemmodell für CEP-Systeme vor, auf das in dieser Arbeit Bezug genommen wird.

Kapitel 4 – Parallelisierungsrahmenwerk: Hierin wird der erste praktische Teil der Diplomarbeit, die Implementierung eines Rahmenwerks für die selektionsbasierte Parallelisierung, beschrieben.

Kapitel 5 – Zielstellung: Nach einer Motivation werden die Ziele definiert, die diese Diplomarbeit im Einzelnen hat.

Kapitel 6 – Problemlösung: Dieses Kapitel enthält die erarbeiteten Lösungen. Zudem beschreibt es die Grundlagen der Warteschlangentheorie, auf die sich die Problemlösung stützt.

Kapitel 7 – Lösungsauswertung: Die zuvor beschriebene Lösung wird anhand von Experimenten ausgewertet und analysiert.

Kapitel 8 – Related Work: Hier werden verschiedene andere Ansätze aus der Wissenschaft und Industrie aufgeführt.

Kapitel 9 – Zusammenfassung und Ausblick: Die Ergebnisse der Arbeit werden kurz zusammengefasst und Anknüpfungspunkte vorgestellt.

2 Einführung in das Complex-Event-Processing

2.1 Complex-Event-Processing

Die Terminologie Complex-Event-Processing, abgekürzt CEP und teilweise auch kurz nur als Event-Processing bezeichnet, hat sowohl seit einiger Zeit in der Wissenschaft als auch in neuerer Zeit in der Industrie an Bedeutung gewonnen. Der CEP-Begriff geht auf David Luckhams Buch *The Power of Events* aus dem Jahr 2002 [Luc02] zurück. Unter Complex-Event-Processing versteht man die Situationserkennung durch das Verarbeiten komplexer Ereignisse in Echtzeit, indem aus kleinen Informationen höherwertige Erkenntnisse abgeleitet werden. Anhand der erkannten Muster auf Strömen niederwertiger Ereignisse wird auf Situationen von Interesse in der umgebenden Welt geschlossen. Beispielsweise erzeugen viele Sensoren kontinuierlich verschiedene Messpunkte, die als Ereignisströme dem CEP-System zugeführt werden. In dieser riesigen Datenmenge sorgt das CEP-System dafür, dass bestimmte Muster in den Messpunkten erkannt oder verschiedene Messwerte miteinander korreliert werden und so zu neuen Ereignissen führen, die in weiteren Schritten auf ähnliche Weise verarbeitet werden können. Dies geschieht beispielsweise durch die Aggregation von Messpunkten, indem Mittel-, Minimal- oder Maximalwerte innerhalb eines Zeitintervalls gebildet werden. Unter Mustererkennung kann man sich vorstellen, dass durch die Beobachtung von fallenden oder sinkenden Temperaturen, schnell verändernden Luftdrücken und -feuchtigkeiten früh Fehler in Produktionsanlagen erkannt werden können. Hierzu müssen unterschiedliche Messwerte miteinander in Verbindung gebracht und unter Einhaltung von Bedingungen durch logische Operatoren wie z.B. *und*, *oder* usw. verknüpft werden. Am Ende erhalten die Kunden des Systems eine gefilterte Sicht, die die komplexeren Aktivitäten im System beschreiben. Daraufhin können die Kunden auf die eingetretene Situation rasch reagieren, indem sie z.B. einen Techniker benachrichtigen, der die Maschine genauer untersuchen und reparieren kann, sodass der Ausfall in Grenzen gehalten wird. Weitere Anwendungsbeispiele sind das automatische Nachbestellen von Bauteilen, wenn der Lagerbestand unter einen Grenzwert fällt oder das Erkennen von verschiedenen Trends in Börsenkursen mit der anschließenden Benachrichtigung von Maklern [Mis91].

CEP-Systeme lassen sich in verschiedenen Ausprägungen finden und bedienen sich diverser Techniken, Methoden und Werkzeuge. Dies liegt mitunter daran, dass sie unterschiedlichen Gebieten entspringen, etwa der Geschäftsprozessüberwachung, dem Bereich der Sensornetze oder der Marktdatenanalyse. Jede Disziplin prägte unabhängig ihre eigenen Ansätze. Auch wird danach differenziert, ob im Voraus festgelegte Muster oder zu Beginn noch völlig unbekannte Muster als komplexe Ereignisse erkannt werden sollen [EB09]. Ein Ereignis kann

prinzipiell alles sein, was geschehen ist oder als geschehen angesehen wird bzw. ein solches Geschehen zum Zwecke der maschinellen Verarbeitung repräsentiert [LS08].

Im Informatiklexikon der Gesellschaft für Informatik ist folgende Kurzbeschreibung zu finden:

> „CEP ist ein Sammelbegriff für Methoden, Techniken und Werkzeuge, um Ereignisse zu verarbeiten während sie passieren, also kontinuierlich und zeitnah. CEP leitet aus Ereignissen höheres, wertvolles Wissen in Form von sog. komplexen Ereignissen, d.h. Situationen[,] die sich nur als Kombination mehrerer Ereignisse erkennen lassen, ab." [EB09]

2.1.1 Abgrenzung zu verwandten Gebieten

Aufgrund der Situationserkennung sollen automatisch Aktionen ausgelöst werden; dies kann die Benachrichtigung eines menschlichen Benutzers sein oder Interaktionen mit beliebigen anderen Anwendungen, Geschäftsprozessen oder auch weiteren CEP-Systemen. Ebenso ist das Ansteuern eines Aktors möglich, etwa das Einschalten einer Sprinkleranlage bei der Branderkennung durch Rauch- und Temperatursensorereignisse. Das CEP-System beschränkt sich jedoch in dieser Arbeit auf das Erkennen von komplexen Ereignissen [EB09].

CEP wird als Vermittlungssoftware betrachtet, daher werden die Ereignisquellen und -senken sowie deren Ereigniserzeugung/-konvertierung und -endverarbeitung nicht dem CEP-System selbst zugerechnet. Insbesondere fällt die Sensortechnik nicht darunter, ebenso wenig wie das alleinige Reagieren auf erkannte Ereignisse, z.B. Ansteuern von Aktorhardware oder Visualisieren von Ergebnissen [May12].

Ein verwandtes Gebiet ist das Stream-Processing. Hierbei lassen sich Datensätze mithilfe von Abfragen aus Teildatenströmen extrahieren, wie dies beispielsweise auch bei Datenbankverwaltungssystemen der Fall ist. Der Fokus liegt jedoch mehr auf dem Abfragen bestimmter relevanter Datensätze, wohingegen das Verarbeiten von einfachen zu komplexen Ereignissen eher in den Hintergrund rückt [May12]. Generell ist die Eingangsdatenrate beim Stream-Processing deutlich höher als beim Event-Processing und kann in großen Schüben erfolgen. Entprechend sind beim Stream-Processing generell mehr Quality-of-Service-Garantien enthalten, wohingegen CEP tendenziell mehr einen Best-Effort-Ansatz verfolgt. Darin begründet liegt auch die stärkere Optimierung auf „unteren" Schichten beim Stream-Processing; so nutzen etwa mehrere Verarbeitungseinheiten gemeinsam eine Eingangswarteschlange oder teilen sich Rechenteilergebnisse [CJ09]. Eine scharfe Abgrenzung zwischen Stream-Processing und Complex-Event-Processing ist jedoch aufgrund der Verwandtheit beider Techniken nicht möglich. Zudem nähern sich über die Zeit hinweg beide Verfahren immer mehr an, da gute Ideen stets wechselseitig übernommen werden.

2.2 Definitionssprachen und Ausführungsmodelle

Die Ereigniskorrelation kann aus zwei unterschiedlichen Grundmechanismen bestehen. So ist es einerseits möglich, dass die zu erkennenden Ereignismuster bereits im Voraus bekannt sind und das CEP-System anhand dieser die komplexen Ereignisse ableitet. In diesem Fall sind je nach Funktionsreichtum des eingesetzten CEP-Systems die Mustererkennungen fest einprogrammiert oder können auch während der Laufzeit mithilfe von speziellen Abfragesprachen verändert werden. Jedoch wird die Erkennung durch das System selbst nicht angepasst. Die Diplomarbeit stützt sich auf diesen Ansatz. Die Alternative ist das Erkennen von anfangs noch unbekannten Mustern. Dies kann beispielsweise durch den Einsatz von maschinellem Lernen erfolgen, wobei dann das CEP-System von sich aus neue Mustererkennungsregeln erstellt oder bestehende anpasst, um dann dadurch die fragliche Situation zu erkennen [May12]. Für den ersten Ansatz existiert bereits eine Menge an Definitionssprachen und auch einige große Softwareunternehmen haben mittlerweile CEP-Systeme im Angebot. Meist arbeiten diese mit Abfragesprachen [Int13, Ora13]. Teilweise sind auch schon exemplarische Implementierungen für den zweiten Ansatz realisiert worden [WASW07].

Als bekannteste Definitionssprache für Ereignismuster ist wohl *snoop* [Mis91] zu nennen, die von Deepak Mishra im Rahmen seiner Masterarbeit entstand und seither weiterentwickelt wurde [CM94]. Sie wird gern im akademischen Umfeld zum Vergleichen von Korrelationsmächtigkeiten verschiedener Systeme verwendet. Neben vielen anderen und ähnlichen Sprachen gibt es mit *StreamSQL* [Str13] eine vorgeschlagene Standardabfragesprache [JMS+08], die sich an SQL anlehnt und mithilfe derer man ebenfalls Korrelationsmuster spezifizieren kann. Sie vereint zwei andere existierende SQL-Spracherweiterungen [CJ09]. Weiterführende Informationen lassen sich aus dem sehr umfangreichen Vergleich *Processing Flows of Information: From Data Stream to Complex Event Processing* von Gianpaolo Cugola und Alessandro Margara [CM12b] entnehmen.

Ein Ausführungsmodell beschreibt die tatsächliche Umsetzung der definierten Korrelationen in der Definitionssprache auf den Ereignisströmen. Je nach Anwendungsfall sind die Ausführungsmodelle unterschiedlich zugeschnitten und optimiert. Zum Beispiel gibt es Modelle für mobile CEP-Systeme und der damit verbundenen Möglichkeit zur Operatorrekonfiguration [KORR12, HLR+13]. Gegenstand der Forschung ist unter anderem auch die Integration in heterogene Umgebungen in der Industrie [SPR09, SKPR10] oder Zugriffsschutzmaßnahmen und Datenschutz in großverteilten CEP-Systemen, wenn Ereignisse über mehrere Netzgrenzen hinweg fließen [SKRR13].

2.3 Aufbau von CEP-Systemen

Ein CEP-System ist in der Diensteschicht der IT-Infrastruktur angesiedelt, und vermittelt daher zwischen Ereignisquellen und -konsumenten. Somit lassen sich beide Komponenten voneinander entkoppeln und die Komplexität der Gesamtanwendung reduzieren. Das CEP-System sorgt dafür, dass Korrelationen auf den Ereignisströmen gebildet werden, komplexe

Ereignisse für die Situationserkennung entstehen usw. Der grundlegende Aufbau von CEP-Systemen ist immer recht ähnlich. Im Folgenden sollen die vier unterschiedlichen Ansätze genauer vorgestellt werden.

2.3.1 Zentralisiertes CEP

In der einfachsten Form von CEP-Systemen wird ein zentralisierter Ansatz verfolgt. Wie Abbildung 2.1 zeigt, werden verschiedene Ereignisquellen an das CEP-System angeschlossen. Dieses wird auf einem Server als ein einzelnes, zentrales Programm ausgeführt, das alle Korrelationsregeln verfügbar hat und anhand dessen die Verarbeitung der Ereignisse aus den Eingangsströmen zu komplexeren Informationen steuert. Die Ereigniskonsumenten erhalten schließlich die höherwertigen Informationen. Beispielsweise sind IBMs *WebSphere Business Events* [Int13] und *Oracle Event Processing* [Ora13] solch zentralisierte Complex-Event-Processing-Systeme auf dem kommerziellen Markt.

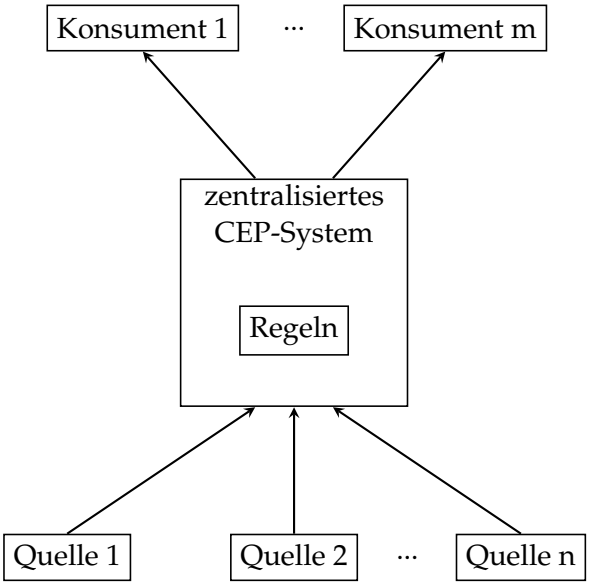


Abbildung 2.1: Aufbau eines zentralisierten CEP-Systems.

Wie die meisten zentralisierten Ansätze haben auch zentralisierte CEP-Systeme eine recht überschaubare Grundkomplexität. Dieser Vorteil spiegelt sich sowohl in der Entwicklung als auch später in der Administration beim Betrieb wider, sodass hier die Kosten vergleichsweise gering ausfallen.

Die Nachteile liegen in der Natur der Sache: schnell kann durch die Zentralisierung ein Flaschenhals oder auch alleiniger Ausfallpunkt entstehen, was die Skalierbarkeit einschränkt. Im Gegensatz zum verteilten Ansatz ist es in diesem Fall häufig auch schwerer bis gar unmöglich, das CEP-System in räumlicher Nähe zu den Ereignisquellen zu platzieren, was bedeutet, dass alle niederwertigen Grundereignisse über längere Strecken transportiert werden müssen. Daher wird einerseits das Netzwerk stärker belastet und es kann zu Verzögerungen kommen,

andererseits sind Quellen und Konsumenten vom CEP-System abhängig, was bedeutet, dass die Korrelationsregeln an eventuell fremder Stelle zentral und vollständig abgelegt werden müssen.

2.3.2 Verteiltes CEP

In verteilten CEP-Systemen wird erstmals versucht, einige Nachteile der zentralisierten Lösungen aufzuheben oder abzuschwächen. So bietet dieser Ansatz durch Hintereinanderschalten von verschiedenen Knoten die Möglichkeit, Ereignisströme stufenweise zu verarbeiten. Statt wie im zentralisierten Fall in einem einzelnen und potentiell sehr großen Schritt aus den Grundereignissen komplexe Ereignisse zu erhalten, wird hier in mehreren und kleineren Schritten vorgegangen. Nach und nach werden verschiedene Ereignisse korreliert und so über verschiedene Zwischenereignisse die endgültigen Ausgangsinformationen erzeugt. Die Operatoren, also die Verarbeitungsknoten, sind hierbei miteinander verbunden und bilden eine Topologie. Bei einigen Anwendungsfällen kann die Gesamtnetzwerklast durch geschickte geografische Anordnung der Operatoren in diesem Ansatz schon annähernd optimal reduziert werden [RDR10, SKR11, OKRR13].

In Abbildung 2.2 ist der Aufbau eines kleinen verteilten CEP-Systems gezeigt. Es besteht aus drei Operatoren, die ihrerseits verschiedene Korrelationsregeln anwenden. Operator 3 erhält die Teilergebnisse von seinen Vorgängeroperatoren 1 und 2, die direkt an die vier Ereignisquellen angeschlossen sind. Nach der hier gezeigten dreistufigen Verarbeitung erhalten die Ereigniskonsumenten die komplexen Ereignisse.

Durch die Verteilung wird einerseits die Skalierbarkeit des Gesamtsystems erhöht, denn die Verarbeitung der Ereignisströme wird auf mehrere Operatoren aufgeteilt. Zudem kann durch die nahe Platzierung der Operatoren an den Quellen die Netzwerklast verringert und somit auch die Verzögerungsszeiten verkürzt sowie die Kosten minimiert werden. Ein weiterer Vorteil ist die Einbringung von eigenen Operatoren ins verteilte System, sodass die interne Korrelationslogik nicht offengelegt werden muss.

Allerdings steigt durch den verteilten Ansatz natürlich die Komplexität, was sich in höheren Kosten in der Entwicklung und im Betrieb bemerkbar macht.

2.3.3 Parallelisiertes CEP

Parallelisierte CEP-Systeme versuchen ebenfalls Unzulänglichkeiten von zentralisierten CEP-Systemen zu beheben. Durch die parallelisierte Verarbeitung von Ereignisströmen werden auch in diesem Fall die Latenzzeiten trotz hohen Ereignisaufkommens verkleinert. Der weitaus wichtigere Vorteil ist jedoch die Erhöhung des Durchsatzes an den Operatoren. Hierzu gibt es verschiedene Ansätze, z.B. wird parallelisierte Hardware wie Grafikkarten verwendet, wobei das zu lösende Grundproblem in mehrere Unterprobleme aufgeteilt wird, die dann parallelisiert gelöst werden [CM12a]. Die Parallelisierung muss jedoch nicht notwendigerweise auf dem gleichen physikalischen Knoten ablaufen, es kann auch über Hostgrenzen hinweg

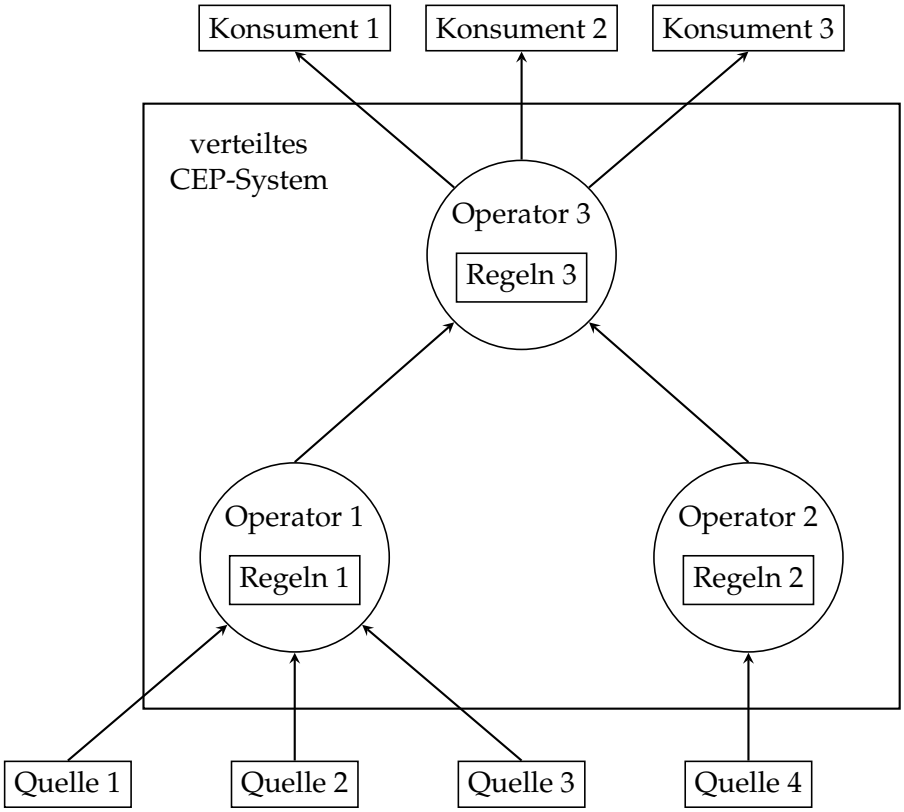


Abbildung 2.2: Aufbau eines verteilten CEP-Systems.

parallel korreliert und gefiltert werden. *Esper* ist ein parallelisiertes Open-Source-CEP-System auf Java- und .NET-Basis [Esp13].

Die Verbesserung der Skalierung ist ein wesentlicher Punkt bei der Parallelisierung. Dazu gibt es zwei grundlegende Arten von Skalierung: die vertikale und die horizontale Skalierung. Bei der Vertikalen werden einem Knoten Ressourcen hinzugefügt, z.B. weiterer Speicherplatz, mehr Rechenkapazität durch eine schnellere CPU usw. Unabhängig von der Software kann hierdurch das System sofort ohne Eingriffe in Quellcode oder Konfiguration verbessert und beschleunigt werden. Allerdings lässt sich dieser Effekt nicht beliebig steigern, da er irgendwann an Grenzen stößt. Wenn etwa schon die am besten ausgerüstete Hardware auf dem Markt verwendet wird, kann diese – zumindest kurzfristig – nicht weiter aufgerüstet werden [MMSW07].

Sofern die Software entsprechend geschrieben ist, kann jedoch bei der horizontalen Skalierung dieser Effekt nahezu beliebig stark gesteigert werden. Hierbei werden dem System neue Knoten hinzugefügt, um die Leistung zu verbessern. Allerdings sind nicht alle Programme gleich gut parallelisierbar, sodass das Hinzunehmen von neuen Rechnern nicht in allen Fällen zur Leistungssteigerung führt [MMSW07].

Parallelisierung kann auch für eine redundante Auslegung der Operatoren genutzt werden. Dies kann im Fehlerfall – beim Ausfall eines Operators – zu einem schnelleren Ausliefern

von komplexen Ereignissen führen, da die Berechnungen von einem anderen Operator auf dem gleichen Teileingangsstrom schon vorliegen. Somit muss dieser Part nicht extra erneut in Auftrag gegeben werden, was zwangsläufig zu einer Verzögerung führen würde. Nachteilig ist jedoch die nicht optimale Nutzung der vorhandenen Systemressourcen im Normalbetrieb.

Bei parallelisierten CEP-Systemen ist ein Operator immer dreigeteilt: ein Splitter teilt am Anfang die Eingangsströme auf und leitet die Ereignisse an verschiedene Operatorinstanzen weiter. Diese verarbeiten die ihnen zugeteilten Ereignisse und schicken ihrerseits ihre produzierten Ergebnisse an eine dritte, sortierende Komponente weiter, die diese in eine eindeutige Reihenfolge bringen und an die Ereigniskonsumenten weiterreichen kann.

2.3.3.1 Zustandsbasierte Operatorparallelisierung

Bei der zustandsbasierten Parallelisierung eines Operators wird die Korrelationsregel in einen endlichen Automaten überführt. Als Annahme liegt zugrunde, dass im Operator die Mustererkennung auf den Ereignisströmen am längsten dauert, diese also parallelisiert werden muss. Abbildung 2.3 zeigt den Aufbau eines Operators. Für jede Zustandsvariable aus dem Automaten (in Abbildung 2.3 sind das *A*, *B* und *C*) ist genau eine Operatorinstanz vorhanden, die versucht, dieses Teil des Musters zu erkennen. Jede Operatorinstanz erhält vom Splitter (*Input Handler*) den kompletten Eingangsstrom, zudem erhalten sie alle die Zwischenergebnisse ihrer Vorgängerinstanzen, um so mit den vorigen Teilergebnissen die Erkennung auf dem Eingangsstrom durchzuführen. Eine Operatorinstanz muss nur dann ihr eigenes Teilmuster überprüfen, wenn ihre Vorgänger Erfolg verkündeten. Somit bleibt eine unnötige Berechnung aus. In diesem Fall ist nur die letzte Operatorinstanz in der Kette, die den Automaten realisiert, mit der sortierenden Komponente, dem *Match Output Handler*, verbunden. Es wird also die Auswertung der Zustandsprädikate parallelisiert. Bei dieser Parallelisierungsart ist die Skalierbarkeit jedoch beschränkt, sie hängt unmittelbar von der Anzahl der Zustandsvariablen des in den Automaten überführten Korrelationsmusters ab. Außerdem können die durch die Unausgewogenheit der damit verbundenen Lastverteilung die zur Verfügung stehenden Ressourcen nicht optimal ausgenutzt werden [BDWT13].

2.3.3.2 Ablaufbasierte Operatorparallelisierung

Wie sich herausstellt, ist die ablaufbasierte Parallelisierung effizienter und skalierungsgeeigneter. In Abbildung 2.4 auf Seite 18 ist der Aufbau dazu dargestellt. Wie auch bei der zustandsbasierten Operatorparallelisierung wird hier das Korrelationsmuster als endlicher Automat umgesetzt. Allerdings sind alle Operatorinstanzen identisch und erkennen das gesamte Muster, nicht nur einen kleinen Teil davon. Somit lassen sich die Vorgänge beschleunigen, da im Negativfall die restlichen Erkennungen auf dem gleichen Knoten verworfen werden können. Im Gegensatz zum anderen Modell werden nachfolgende Instanzen nicht unnötig damit aufgehalten. Der Eingangsstrom wird bei diesem Verfahren in verschiedene Ereignisbündel aufgeteilt. Diese überlappen sich und sind groß genug, sodass ein komplettes zu erkennendes Ereignismuster in mindestens einem der Ereignisbündel liegt. Jede Operatorinstanz arbeitet

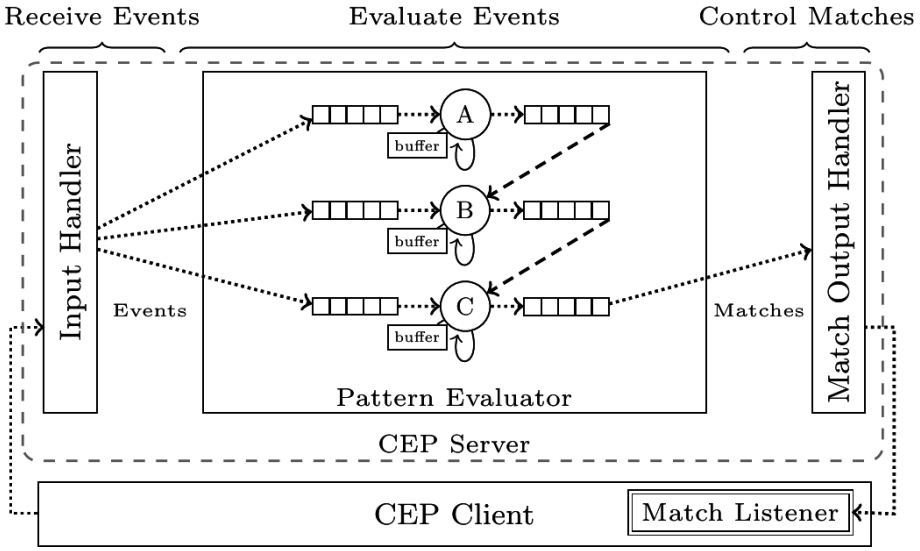


Abbildung 2.3: Zustandsbasierte Parallelisierung eines Operators [BDWT13].

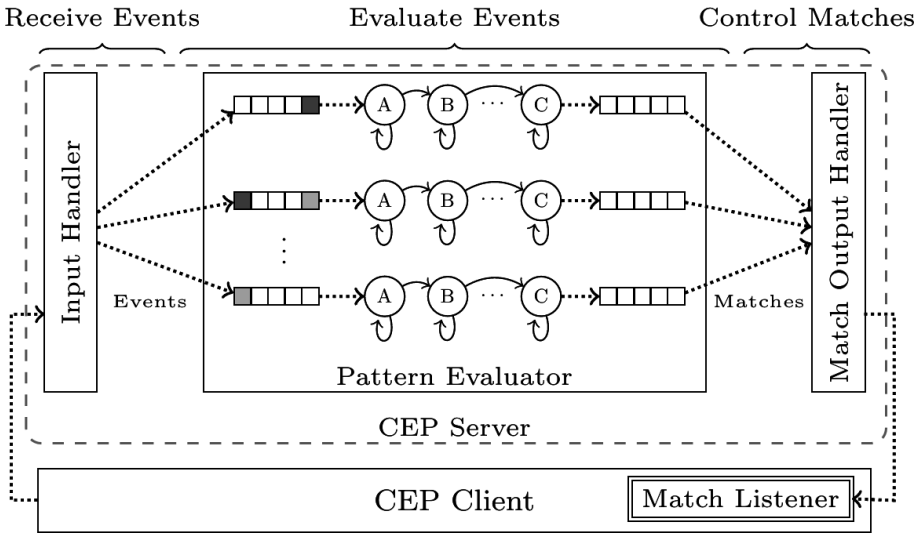


Abbildung 2.4: Ablaufbasierte Parallelisierung eines Operators [BDWT13].

auf einem der Bündel. Da gewährleistet ist, dass die Teilströme groß genug sind und das Muster in mindestens einem davon vollständig enthalten ist, wird es auch zuverlässig erkannt. Dies funktioniert aber nur im Falle von endlich langen Mustern sehr zuverlässig, bei beliebig groß werdenden Mustern kann die zu bearbeitende, überlappende Teileingangsstromgröße nur schwer abgeschätzt werden und es müssen Einschränkungen hingenommen werden. In Abbildung 2.4 sind gleiche Ereignisse in den verschiedenen Eingangswarteschlangen der Operatoren mit dem gleichen Grauton eingefärbt. Mit diesem Ansatz wird die Ereignisreplikation reduziert und somit das Netzwerk etwas entlastet. Zudem ist die Skalierbarkeit vom Korrelationsmuster unabhängig [BDWT13].

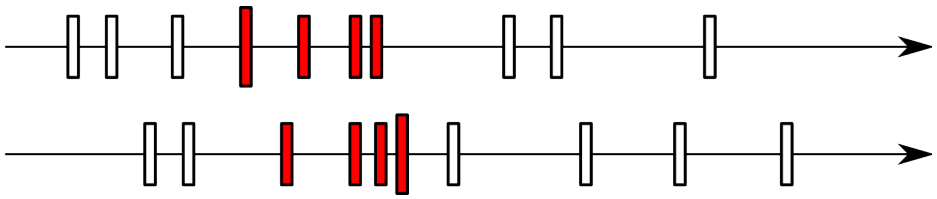


Abbildung 2.5: Zwei Eingabeströme eines Operators, auf denen die roten Ereignisse in einer Selektion korreliert sind. Die beiden höher dargestellten Ereignisse bilden hierbei den Anfang bzw. das Ende der Selektion.

2.3.3.3 Selektionsbasierte Operatorparallelisierung

Ein anderer Ansatz ist die selektionsbasierte Operatorparallelisierung. Hierbei werden Ereignisse in den Eingangsströmen zusammen gruppiert, sogenannte Selektionen darauf gebildet. Eine Selektion ist genau die Menge von Ereignissen, die in einem Korrelationsschritt von einer Operatorinstanz auf die ausgehenden komplexen Ereignisse abgebildet wird. Dieses Verfahren baut auf einem allgemeinen Operatormodell auf, das den Eingangsstrom schrittweise verarbeiet. Ein Prädikat erkennt den Beginn einer Selektion, ein optionales Zweites das Ende. Sequenzen ohne Ende werden in dieser Arbeit nicht behandelt. Alle Ereignisse, die zwischen den von dem Prädikat erkannten Start- und Endereignissen liegen, sind hierbei in der Selektion enthalten, siehe Abbildung 2.5. Ein Ereignis kann in beliebig vielen Selektionen enthalten sein, so auch in gar keiner oder sogar in mehreren. Im letzteren Fall sind die Selektionen überlappend. Mithilfe der Selektionen ist es möglich, die parallelisierte Verarbeitung einfacher und effizienter aufzuteilen, da die Selektionsbildung explizit für jeden Anwendungsfall neu angegeben werden muss. Damit lassen sich der jeweiligen Korrelation entsprechend sehr günstige Ereignisteilströme bilden, die auf das Muster hin untersucht werden sollen [May13b].

Einer Operatorinstanz wird vom Splitter eine komplette Selektion überreicht. Diese kann daraufhin den Teilstrom nach den spezifizierten Mustern durchsuchen. Ergebnisse meldet die Operatorinstanz an den Sequenznummerngenerator, der die eintreffenden komplexen Ereignisse eindeutig sortiert und an die Konsumenten ausliefert. Dieses Vorgehen wird auch in dieser Diplomarbeit angewendet.

2.3.4 Parallelisiertes und verteiltes CEP

Wie der Name vermuten lässt, kombinieren parallelisierte und verteilte CEP-Systeme die Eigenschaften der beiden zuvor beschriebenen Gattungen. Die Mustererkennung an den Ereignisströmen erfolgt dabei schrittweise über mehrere Operatoren, wobei die Operatoren selbst parallelisiert sind. Der Aufbau eines parallelisierten und verteilten CEP-Systems ist in Abbildung 2.6 auf Seite 20 gezeigt. Vier Ereignisquellen sind an zwei unterschiedliche Operatoren angeschlossen. Jeder der gezeigten Operatoren hat zwei Operatorinstanzen, die die Verarbeitung der Ereignisse übernehmen. In einem zweiten Schritt werden die Zwischenergebnisse der Operatoren 1 und 2 dem dritten Operator eingespeist, der diese seinen beiden

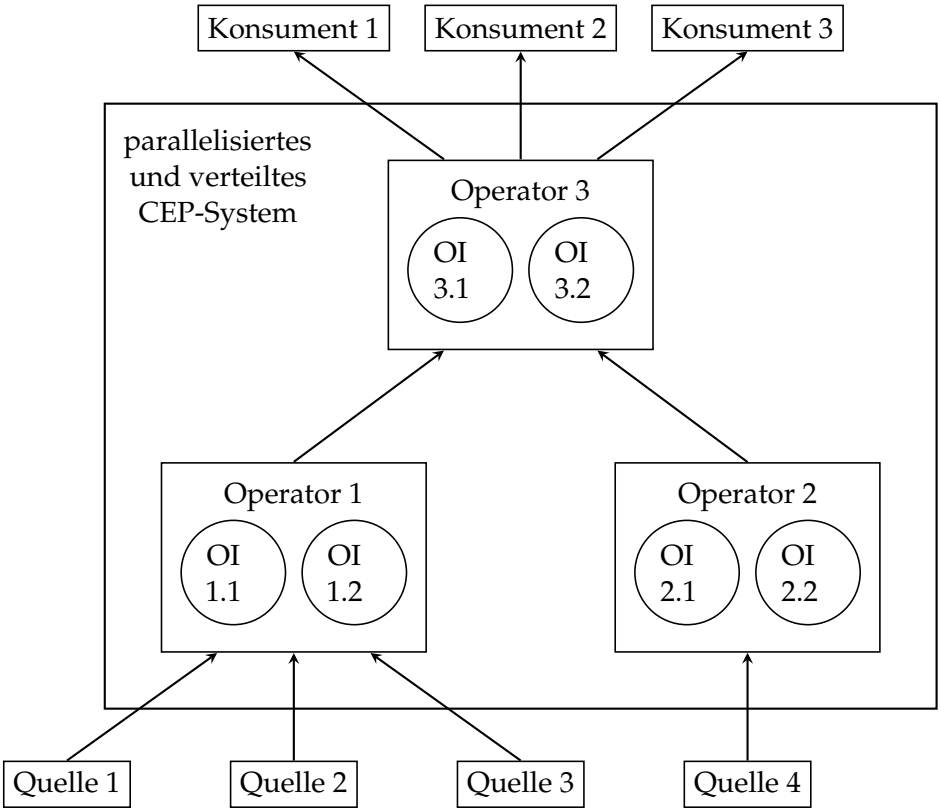


Abbildung 2.6: Aufbau eines parallelisierten und verteilten CEP-Systems.

Operatorinstanzen 3.1 und 3.2 zur Korrelierung und Filterung gibt. Schließlich verlassen die komplexen Ereignisse das System und erreichen die drei verbundenen Konsumenten. Dies ist die mächtigste und zugleich auch komplizierteste Aufbauvariante eines Complex-Event-Processing-Systems.

2.3.5 Fazit

Für Anwendungsfälle mit recht geringem Netzwerkverkehrsaufkommen an den Ereignisquellen ist ein zentrales CEP-System ausreichend. In vielen Szenarien ist dies jedoch nicht gegeben. Bei vielen Basisereignissen in kurzer Zeit, womöglich noch über einen großen Raum verteilt, eignen sich selbst nur-parallelisierte oder nur-verteilte CEP-Systeme nicht mehr und es muss der Ansatz des verteilten und parallelisierten CEP-Systems gewählt werden. Sie bieten trotz ihrer Komplexität den größten Handlungsspielraum. Diese Diplomarbeit stützt sich daher fortan ausschließlich auf diese Form und verwendet dabei die selektionsbasierte Operatorparallelisierung in Verbindung mit hintereinandergeschalteten Operatoren aus dem verteilten Ansatz.

2.4 Anforderungen

An CEP-Systeme werden verschiedene Anforderungen gestellt, die sie erfüllen müssen. Eine Anforderung wird als *hart* bezeichnet, wenn sie unbedingt eingehalten werden muss. Eine *weiche* Anforderung hingegen darf hin und wieder auch kurzzeitig verletzt werden. Generell werden jedoch fast nur harte Anforderungen an CEP-Systeme gestellt. Teilweise sind aber einige nur schwer miteinander in Einklang zu bringen.

2.4.1 Korrektheit

Korrektheit heißt, dass alle erkannten Situationen tatsächlich auch in der echten Welt eintraten, also dass insbesondere im Fehlerfall keine falschen komplexen Ereignisse erzeugt werden. Jede Abweichung vom Normalbetrieb führt dabei in den Fehlerfall, dies kann durch Ausfälle von Rechenknoten oder Kommunikationsverbindungen geschehen. Während eines Ausfalls kann es vorkommen, dass dadurch benötigte Teilergebnisse an späterer Stelle fehlen und es somit dort zu falschen Schlüssen kommt. Bei falschen Ereignissen unterscheidet man zwischen *falsch-positiv* und *falsch-negativ*. Falsch-positiv bedeutet, dass ein komplexes Ereignis erzeugt und an den Konsumenten ausgeliefert wird, obwohl dies im Normalfall nicht passiert wäre. In diesem Fall wurde also fälschlicherweise ein Muster erkannt, das so gar nicht in den Ereigniseingangsströmen vorhanden ist. Bei den falsch-negativen Ereignissen wird normalerweise eine Situation erkannt, im Fehlerfall geschieht dies jedoch nicht, die Mustererkennung schlägt fehl und den Konsumenten wird ein komplexes Ereignis vorenthalten. Korrektheit ist immer im Zusammenhang mit Ausfällen zu sehen, wenn sich also das System nicht im Normalbetrieb befindet. Sie meint explizit nicht Modellfehler.

Häufig ist Korrektheit eine harte Anforderung, sie kann teilweise aber auch als abgeschwächte Form an eine Wahrscheinlichkeit gekoppelt sein, sodass beispielsweise die ausgelieferten Ereignisse zu 99 % korrekt sind. Je nach Anwendungsfall ist dies sehr unterschiedlich. Z.B. könnte eine Chemiewerküberwachung definieren, dass zwar im gewissen Rahmen Fehlalarme ausgelöst werden dürfen (falsch-positive Ereignisse wären somit hin und wieder in Ordnung), es aber unter keinen Umständen dazu kommen darf, dass beim CEP-Versagen im Ernstfall keine Gefahr erkannt wird (falsch-negative Ereignisse müssen ausgeschlossen werden).

In dieser Arbeit ist die Korrektheit eine harte Anforderung, die zu 100 % in allen Situationen gegeben sein muss.

2.4.2 Echtzeit/Rechtzeitigkeit

CEP-Systeme sind generell Echtzeitsysteme, das bedeutet, dass bestimmte Fristen eingehalten werden müssen. So muss innerhalb einer vorgegebenen Maximalzeit eine Situation erkannt worden sein, da die Information sonst im besten Falle wertlos ist. Bei kritischeren Systemen, z.B. der Überwachung einer Chemieanlage, kann die zu späte Erkennung eines Lecks oder Entstehung eines gefährlichen Gasgemisches fatale Folgen haben, sodass eine rechtzeitige Reaktion zur Gefahrenabwehr nicht erfolgen kann.

Nicht selten ist im konkreten Fall die Rechtzeitigkeits- mit der Korrektheitsanforderung schwer zu vereinen. Nichtsdestotrotz können beide zusammen erfüllt werden, dies geschieht dann meist unter Einschränkung der Systemleistung.

Diese Arbeit hat ebenfalls zum Ziel, Rechtzeitigkeit zu garantieren.

2.4.3 Systemleistung/Effizienz

Als Betreiber eines CEP-Systems möchte man sowohl möglichst gut ausgelastete Operatoren als auch ein so wenig wie möglich belastetes Netzwerk haben. Jedoch droht aufgrund stark ausgelasteter Knoten im Fehlerfall schnell eine Überlast, denn durch den Wegfall einer Komponente müssen die verbleibenden Einheiten mehr Aufträge verarbeiten. Daher muss schnell für Ersatz gesorgt werden, was aber nicht immer sofort geschehen kann, da z.B. erst ein neuer Knoten provisioniert und anschließend auf diesem ein Operator ausgebracht werden muss. Steht eine Ersatzkomponente dauerhaft in Bereitschaft, kann zwar schnell auf Ausfälle reagiert werden, doch führt dies unweigerlich zu einer verminderten Ressourceneffizienz. Auch die Alternative, mittels redundanter Ereignisverarbeitung eintretende Fehler abfangen zu können, senkt die Leistung des Systems, da sowohl zusätzliche Operatoren benötigt werden als auch das Netzwerk durch die mehrfache Verteilung stärker beansprucht wird.

Wie in vielen Fällen, so wird auch in dieser Arbeit daher eine eingeschränkte Systemleistung gegen Echtzeit und Korrektheit „eingetauscht“.

3 Systemmodell zu verteilten und parallelisierten CEP-Systemen

In diesem Kapitel führe ich das Systemmodell des parallelisierten und verteilten Complex-Event-Processing-Systems ein, das auch im praktischen Teil zur Anwendung kommt.

3.1 Operatorgraph

Ein parallelisiertes und verteiltes CEP-System kann durch einen gerichteten Operatorgraphen $G(\Omega, E)$ dargestellt werden. Hierbei ist $\Omega = \{\omega_1, \omega_2, \cdots, \omega_n\}$ eine Menge von parallelisierten Operatoren. Diese sind miteinander unidirektional durch Kommunikationskanäle verbunden, was in der Kantenmenge $E \subseteq \Omega \times \Omega$ kodiert ist. Schleifen sind dabei nicht zulässig. Ein Operator ω_i besteht seinerseits aus mehreren Operatorinstanzen ω_i^j, die die eigentliche Korrelationsarbeit leisten. Die über die verschiedenen Eingangsströme eingehende Ereignisse werden von den Operatoren ω_i serialisiert, also in eine eindeutige Reihenfolge gebracht. Start- und Endprädikate erzeugen auf dem serialisierten Strom Selektionen, die vom Splitter schließlich als komplette Einheit an die Operatorinstanzen zur Korrelation und Filterung über weitere Komunikationskanäle gesendet werden. Dort kann eine Korrelationsfunktion f ausgeführt werden, die vom Eingangsstrom I_i^j in den Ausgangsstrom O_i^j abbildet: $f_i^j : I_i^j \mapsto O_i^j$. Einzelne Eingangsströme werden als I_{i_k} eines Operators ω_i bezeichnet und führen nach dem Zusammenführen am Splitter in den Gesamteingangsstrom I_i. Eine Teilmenge dieses Stroms I_i wird dann an die Operatorinstanzen weitergeleitet und steht dort als I_i^j zur Verfügung. Der Ausgangsstrom O_i^j der Operatorinstanz ω_i^j führt über den Sequenznummerngenerator des Operators ω_i in dessen Gesamtausgangsstrom O_i. Abbildung 3.1 zeigt den Operator ω_1 mit allen relevanten Strömen.

3.2 Ereignisse

Ein Ereignis σ_i ist ein konkretes Datenobjekt von einem bestimmten Datentyp. Neben den eigentlichen Nutzdaten enthält es auch verschiedene Metainformationen. Hierunter fallen ein unveränderlicher Zeitstempel $\tau(\sigma_i)$ sowie eine von diesem Erzeuger eindeutig vergebene Sequenznummer $\rho(\sigma_i)$, die ebenfalls nicht verändert werden kann.

Ereignisse sind die Informationen oder Daten, mit denen ein CEP-System arbeitet. Sie kapseln alle benötigten Informationen, die zum Verarbeiten im System benötigt werden. Sowohl

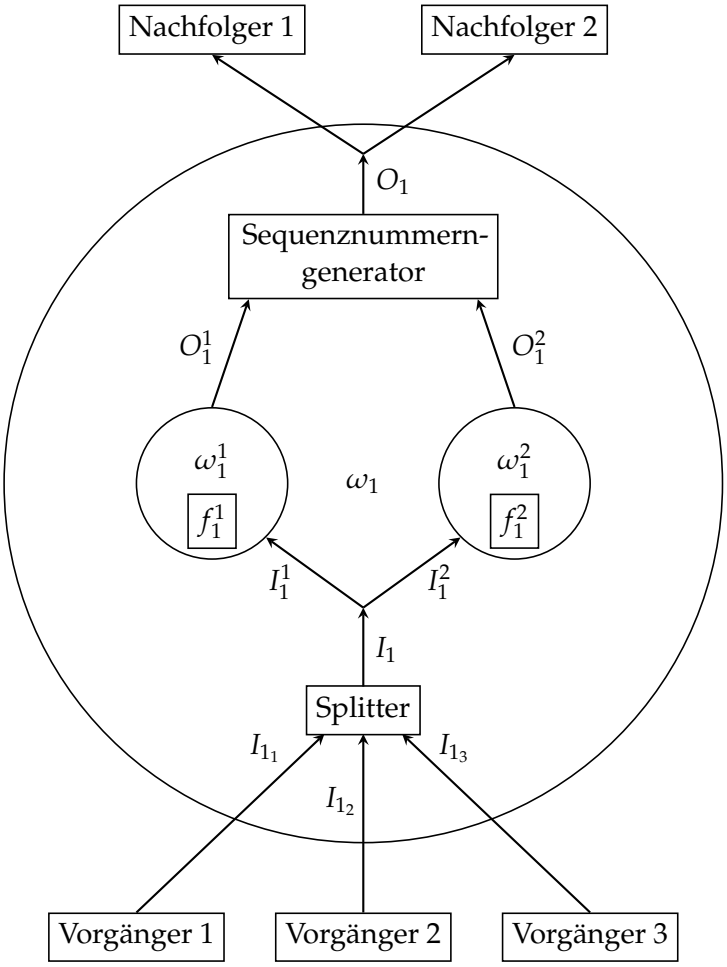


Abbildung 3.1: Ereignisströme eines Operators ω_1 mit zwei Operatorinstanzen ω_1^1 und ω_1^2 aus einem verteilten und parallelisierten CEP-System. Die drei Vorgänger können hierbei entweder andere Operatoren oder Ereignisquellen sein, die zwei Nachfolger Folgeoperatoren oder Ereigniskonsumenten.

innerhalb – zwischen den Operatoren und den Instanzen – als auch außerhalb des CEP-Systems – von der Quelle zum ersten Operator sowie vom letzten Operator zum Konsumenten – werden Ereignisse zum Datenaustausch versendet und empfangen.

Der **Ereignistyp** ist für die Ausgangsströme der Ereignisproduzenten eindeutig festgelegt und technisch als Zeichenkette kodiert. Somit kann anhand des Typs der Produzent des Ereignisses eindeutig festgestellt werden.

Der **Zeitstempel** τ hält entweder den Zeitpunkt oder die Zeitspanne fest, wann ein Ereignis eingetreten ist. Dies hängt vom konkreten Anwendungsfall ab. In der Regel beschreibt der Zeitstempel nicht den Zeitpunkt der Ereigniserzeugung oder Situationserkennung an den Operatorinstanzen sondern den des tatsächlichen Auftritt des Ereignisses. Somit kann die Konsumentenanwendung feststellen, wann eine Situation wirklich eingetreten ist. Die

Verarbeitungsgeschwindigkeit unterliegt ohnehin gewissen Schwankungen, sodass die Erzeugungszeit hier nicht reproduzierbar und damit auch für die Konsumenten nicht weiter hilfreich wäre. Ein von einer Operatorinstanz erzeugtes komplexeres Ereignis wird daher in Abhängigkeit der eingehenden Ereignisse mit korrelierenden Zeitstempeln versehen. In den meisten Fällen wird hierbei eine Zeitspanne als Zeitstempel gewählt, die sich aus den Zeitstempeln des ersten und des letzten korrelierten Ereignisses zusammensetzt. Wenn ein Ereignis Teil mehrerer Korrelationen ist, können daher mehrere unterschiedliche Ereignisse desselben Typs den gleichen Zeitstempel aufweisen. Es ist nicht zwingend erforderlich, dass der Zeitstempel die Realzeit enthält, es kann auch eine beliebige andere Zeitangabe verwendet werden, solange alle Ereignisquellen hinreichend genau synchronisiert sind und dieselbe Zeitauffassung haben. Technisch kann der Zeitstempel beispielsweise eine Ganzzahl oder Fließkommazahl sein, die beispielsweise den Unixzeitstempel als Echtzeit verwendet. Als weitere Alternative könnte eine Zeichenkette formatiert nach ISO 8601 bzw. RFC 3339 [KN02] genutzt werden. Ein Tupel $(Start, Ende)$ mit denselben Eigenschaften für $Start$ und $Ende$ wäre genauso gut denkbar.

Die **Sequenznummer** ρ dient zur logischen Ordnung verschiedener Ereignisse in einem Ereignisstrom eines Produzenten. Sequenznummern werden streng monoton steigend vergeben, sodass eine totale Ordnung bezüglich eines Produzenten möglich ist. Dies bedeutet, dass eine kleinere Sequenznummer $\rho(\sigma_i)$ des Ereignisses σ_i gleichbedeutend ist mit der Tatsache, dass das Ereignis σ_j mit der größeren Sequenznummer $\rho(\sigma_j)$ zeitlich nach σ_i erzeugt wurde, sofern sowohl σ_i als auch σ_j vom selben Produzenten stammen. Somit können Ereignisse eines Erzeugers eindeutig in eine Reihenfolge gebracht werden. Über mehrere Produzenten hinweg lässt sich jedoch mit der Sequenznummer allein keine Ereignisordnung herstellen. Die Sequenznummer ist technisch betrachtet eine natürliche Ganzzahl, kann aber teilweise auch ein Tupel sein, um hierarchische Informationen festzuhalten (siehe Abschnitt 3.3.5 *Sequenzierung der Ausgangsströme*).

Die **Nutzdaten** eines Ereignisses sind die erforderlichen Attribute, die zur Weiterverarbeitung des Ereignisses notwendig sind. Sie enthalten alle Informationen, die von Interesse sind. Technisch kann dies als einfache Menge von Schlüssel-Werte-Paaren oder auch als verschachtelte Variante realisiert sein.

3.3 Operatoren und Operatorinstanzen

Ein **Operator** ω_i ist eine logische Einheit, bestehend aus einem Splitter, einem Sequenznummerngenerator sowie mehreren Operatorinstanzen. Nach außen hin sieht ein Operator transparent wie eine einzelne Komponente aus. Er erhält mehrere Eingangsströme I_{i_k} von seinen Vorgängeroperatoren oder direkt von den Ereignisquellen, die er mithilfe des Splitters zu einem Gesamteingangsstrom I_i zusammenführt. Die Ausgabe erfolgt auf seinem Ereignisausgangsstrom O_i, der wiederum seinen Nachfolgeroperatoren oder direkt den Ereigniskonsumenten als Eingang dient.

Intern verwaltet der Operator ω_i mehrere **Operatorinstanzen** ω_i^j, die die Korrelationsfunktion $f_i^j : I_i^j \mapsto O_i^j$ ausführen. Hierbei sind die Eingangsströme I_i^j der Instanzen an den Gesamteingangsstrom I_i des Operators angebunden. Über die Eingangsströme I_i^j erhalten die Operatorinstanzen ω_i^j Teilmengen der Ereignisse aus dem Operatoreingangsstrom I_i. Diese Teilmengen werden durch Selektionen bestimmt, welche wiederum von der Selektionsschnittstelle im Splitter erzeugt werden. Hierbei muss eine Selektion so aufgebaut sein, dass bei der parallelen Selektionsverarbeitung genau die gleichen Ergebnisse erzeugt werden, wie wenn der Gesamteingangsstrom I_i sequentiell verarbeitet werden würde. Eine Operatorinstanz erhält hierbei immer eine komplette Selektion. Die erzeugten Ereignisse haben dabei einen konkreten Typ. Sinn der Operatorinstanzen ist die parallelisierte Verarbeitung des Gesamteingabestroms I_i am Operator ω_i. Die Ausgabeströme O_i^j der Instanzen münden in den Sequenznummerngenerator, der sie zu einem Gesamtausgabestrom O_i des Operators ω_i serialisiert.

Um die Operatorinstanzen von der Verwaltungsarbeit der Selektionen abzukapseln, werden diese in **Laufzeitumgebungen** ausgeführt. Ein anderes Ziel des Parallelisierungsansatzes ist die Idee, dass bestehende konventionelle Operatoren eines verteilten CEP-Systems recht einfach ohne größere Anpassungen in dieses parallelisierte und verteilte CEP-System portiert werden können. Dazu sollte die Operatorinstanz nichts von Selektionen wissen müssen. Genau genommen sendet der Operator ω_i die zu verarbeitenden Selektionen deshalb auch nicht direkt an die Operatorinstanz ω_i^j sondern an die dafür verantwortliche Laufzeitumgebung, die die darin enthaltenen Ereignisse an ω_i^j weiterreicht. Auch liegt zwischen dem Ausgangsstrom O_i^j und dem Sequenznummerngenerator des Operators ω_i noch die Laufzeitumgebung. Laufzeitumgebungen setzen ihre Operatorinstanzen entsprechend für die zu bearbeitenden Selektionen auf und beenden sie auch zwischendurch, wenn keine zusammenhängenden Selektionen verarbeitet werden sollen. Die Fernsteuerung durch die Laufzeitumgebung ist für die Operatorinstanz jedoch nicht sichtbar.

3.3.1 Operatorenvernetzung

Die Operatoren sind durch Kommunikationsverbindungen vernetzt, über die Ereignisse übertragen werden. Hierbei wird der Ausgangsstrom O_i eines Operators ω_i zu einem von potentiell mehreren Eingangsströmen I_{j_k} des Operators ω_j. Somit ist ω_i ein Vorgänger von ω_j bzw. ω_j ein Nachfolger von ω_i. $pred(\omega_j)$ ist die Menge aller Vorgänger von ω_j, in der also auch ω_i enthalten ist. ω_j ist Teil von $succ(\omega_i)$, der Nachfolgermenge von ω_i. Über beide Mengen lassen sich transitive Hüllen $pred^*(\omega_j)$ bilden, die *alle* Vorgänger – nicht nur die direkten – enthält sowie $succ^*(\omega_i)$ mit allen direkten und indirekten Nachfolgern des Operators ω_i.

3.3.2 Sequenzierung der Eingangsströme

Ein Operator ω_i hat nicht selten mehrere Vorgänger im Operatorgraphen, sei es in Form von anderen Operatoren oder den Ereignisquellen direkt: $|pred(\omega_i)| = v$ wobei $v \geq 1$. Somit gibt

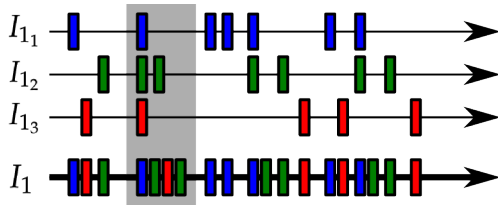


Abbildung 3.2: Serialisierung der Ereignisse aus drei Eingangsströmen I_{1_1}, I_{1_2} und I_{1_3} zum Gesamteingangsstrom I_1 unten. Im grau hervorgehobenen Zeitabschnitt treffen drei Ereignisse gleichzeitig ein, die dann entsprechend der Prozesskennung eindeutig topologisch sortiert werden. Das in dieser Zeitspanne zweite grüne Ereignis aus I_{1_2} folgt im serialisierten Strom I_1 dem ersten roten aus I_{1_3}.

es v Eingangsströme I_{i_1}, I_{i_2}, … I_{i_v}, die zu einem einzelnen Eingangsstrom I_i am Operator zusammengeführt werden müssen, vgl. Abbildung 3.2 mit $v = 3$.

Der Splitter eines Operators ω_i serialisiert zuerst die über die Eingangskanäle I_{i_k} eintreffenden Ereignisse σ_{i_k}. Innerhalb eines einzelnen Eingangsstroms I_{i_k} können sie anhand der aufsteigenden Sequenznummern $\rho(\sigma_{i_k})$ sortiert werden, sofern der Kommunikationskanal dies nicht bereits zusichert, was z.B. bei UDP nicht der Fall wäre. In der Implementierung wird jedoch TCP eingesetzt, somit entfällt diese Notwendigkeit. Nachdem alle Ströme O_{i_k} in sich zeitlich sortiert sind, können die Ereignisse σ_{i_k} aus den verschiedenen Strömen durch Vergleichen der Ereigniszeitstempel $\tau(\sigma_{i_k})$ in eine eindeutige Reihenfolge gebracht werden. Bei Tupeln wird der Anfangszeitpunkt verglichen. Sind die Zeitstempel von zwei oder mehr Ereignissen aus verschiedenen Eingangsströmen identisch, wird ein zweites Kriterium zur Sortierung herangezogen. Dies sind die Prozesskennungen der jeweiligen Vorgänger der beteiligten Eingabeströme. Diese müssen auf jeden Fall unterschiedlich sein. Alternativ kann auch als sekundäres Unterscheidungsmerkmal der Ereignistyp herangezogen werden. Dieser ist gemäß der oben stehenden Definition eindeutig.

3.3.3 Selektionsbildung

Die Grundidee hinter den Operatorinstanzen ist die parallelisierte Verarbeitung der Ereignisse. Hierbei sollen unterschiedliche Ausschnitte aus dem Eingangsstrom von mehreren Instanzen zeitgleich auf vorhandene Muster überprüft werden. Die automatische und effiziente Aufteilung erfolgt durch Selektionen, die durch Prädikate bestimmt wird. Beide Prädikate zum Selektionsbeginn und -ende werden dazu vom Anwender bereitgestellt. Selektionen können nur bei Kenntnis der auf den Operatorinstanzen ausgeführten Überführungsfunktion f Ereignisströme so aufteilen, dass Korrelationsmuster dort mit f auch erkannt werden können.

Eine Selektion ς_i ist eine nichtleere Menge an Ereignissen und enthält hierbei mindestens ein Start- und ein Endereignis, wobei beide auch identisch sein können. In diesem Fall umfasst die Selektion dann lediglich ein Ereignis. Eine Selektion enthält alle Ereignisse zwischen inklusive dem Start- und inklusive dem Endereignis des serialisierten Gesamteingangsstroms I_i. Im

allgemeinen Fall können sich Selektionen auch überlappen, sodass Ereignisse Teil mehrerer Selektionen sind. Dies ist jedoch in diesem Systemmodell nicht der Fall, ein Ereignis kann hier höchstens in einer Selektion enthalten sein. Somit kann es auch Ereignisse geben, die zu keiner Selektion gehören. Selektionen müssen die Anforderung erfüllen, unabhängig voneinander verarbeitbar zu sein und zu einem konsistenten Ausgangsstrom zu führen.

Auf dem serialisierten Eingangsstrom I_i wird von einer Selektionsschnittstelle aus jedes Ereignis auf die Zugehörigkeit zu einer neuen Selektion überprüft. Zudem findet ein Test statt, ob das Ereignis eine bereits gestartete Selektion abschließt. Dies geschieht mithilfe von zwei Prädikaten der Form $Ps(\sigma_i)$ bzw. $Pc(\sigma_i, \varsigma_i)$. Das einstellige Prädikat $Ps(\sigma_i)$ überprüft, ob das Ereignis σ_i eine neue Selektion ς_i einleitet, es also das Startereignis für diese Selektion ist. Mit dem zweistelligen $Pc(\sigma_i, \varsigma_i)$ wird ermittelt, ob das Ereignis σ_i die geöffnete Selektion ς_i schließt, es also das letzte Ereignis dieser Selektion ist. Anhand der Selektionsbildung findet somit eine Vorfilterung der Ereignisse statt. Ein in den Splitter eingebautes Steuerprogramm verteilt schließlich die Selektionen auf die verfügbaren Operatorinstanzen zur eigentlichen Verarbeitung. Die Vergabe kann einfach über eine Ringverteilung geschehen oder auch kompliziertere Heuristiken verwenden, wie etwa eine vermutete Operatorinstanzauslastung.

3.3.4 Ereigniskonsum

Den Operatorinstanzen ist es zur Verarbeitung ihrer zugeteilten Selektionen möglich, Ereignisse beliebig oft in ihre Berechnungen mit einzubeziehen, sofern sie dabei Selektionsgrenzen nicht überschreiten. Schließlich kann eine Operatorinstanz jedoch ein verwendetes Ereignis auch von der weiteren Nutzung in ihren Berechnungen ausschließen und es aus dem Eingangsstrom entfernen. In diesem Fall spricht man von einem *Ereigniskonsum*, d.h. es kann später nicht mehr zu Berechnungen herangezogen werden.

Der Ereigniskonsum kann in CEP-Systemen elementar sein. Oftmals sind Regeln wie „korreliere immer ein Ereignis vom Typ A mit einem vom Typ B“ im System, die für sich alleine teilweise keine eindeutige Verarbeitung zulassen. Liegen mehrere Ereignisse der Typen A und B im Eingangsstrom vor, kann die Operatorinstanz manchmal nicht eindeutig entscheiden, welche der verschiedenen Kombinationen aus A- und B-Ereignissen miteinander korreliert werden sollen. In der Definitionssprache *snoop* [CM94] sind daher Parameterkontexte vorhanden, um eindeutige Korrelationsvorschriften definieren zu können. Da diese in den Operatorinstanzen nachgebildet werden können sollen, ist es nötig, gezielt Ereignisse zu konsumieren. Snoops Parameterkontexte sind im Folgenden kurz erläutert:

Recent: Es werden immer die aktuellsten Ereignisse für eine mögliche Korrelation verwendet und anschließend konsumiert.

Chronicle: Die Korrelation wird in chronologischer Reihenfolge vollzogen, d.h. immer die ältesten Ereignisse korreliert und schließlich konsumiert.

Continous: Jedes vorn im Eingabestrom liegende und in Frage kommende Ereignis wird mit den nachfolgenden ältesten Ereignissen kontinuierlich korreliert. Nur das Startereignis wird konsumiert, da Ereignisse hier nur einmal eine Korrelation beginnen können.

Cumulative: Alle Ereignisse, die zwischen dem ältest möglichen Start- und jüngsten Ereignis liegen, werden korreliert und danach konsumiert [CM94].

3.3.5 Sequenzierung der Ausgangsströme

Am Ende müssen die erzeugten Ereignisse aus den verschiedenen Operatorinstanzen ω_i^j vom Sequenznummerngenerator noch in eine eindeutige Reihenfolge gebracht werden, bevor sie den Operator ω_i durch den Ausgabestrom O_i verlassen. Dies geschieht anhand der von den Laufzeitumgebungen vergebenen Sequenznummertupeln. Die Tupel bestehen aus der aktuellen Selektionskennung sowie einen für jede Selektion neugestarteten Zähler. So können die erzeugten komplexen Ereignisse nach Selektion – und damit Zeit – sortiert in den Gesamtausgangsstrom des Operators ω_i eindeutig und reproduzierbar übernommen werden. Da die Selektionsinformation für den Nachfolgeroperator irrelevant ist – dieser erzeugt neue und andere Selektionen – wird die Sequenznummer vom Splitter in eine fortlaufende Ganzzahl umgeschrieben. Auch die Senke benötigt die internen Verwaltungsinformationen des CEP-Systems nicht.

3.3.6 Fehlermodell

Wie eingangs beschrieben, bilden Operatoren logische Einheiten. Alle Bestandteile (Splitter, Sequenznummerngenerator sowie Operatorinstanzen inklusive den Laufzeitumgebungen) können dabei komplett oder teilweise auf dem gleichen virtuellen oder physikalischen Knoten ausgeführt werden. Nicht selten sind alle Komponenten in eigenen virtuellen Hosts untergebracht, wobei sich einige hiervon auf demselben physikalischen Gerät befinden.

Splitter, Sequenznummerngeneratoren sowie Laufzeitumgebungen sind in diesem Systemmodell fehlerfrei und können nicht abstürzen, sie arbeiten immer korrekt. Operatorinstanzen sind die einzigen Komponenten, die ausfallen können. Sie arbeiten nach dem Crash-Stop-Modell, wonach eine Operatorinstanz beim Absturz alle ihre nichtpersistenten Daten verliert. Zudem hält der Prozess an, es werden keine Daten mehr empfangen und versendet. Andere Operatorinstanzen bekommen hiervon nichts mit. Auch Splitter und Sequenznummerngenerator werden vom Ausfall der Instanz nicht unmittelbar automatisch in Kenntniss gesetzt. Der Sequenznummerngenerator kann jedoch anhand von nicht mehr eintreffenden Ereignissen innerhalb einer bestimmten Zeit den Operatorinstanzausfall feststellen und wird dann auch den Splitter informieren. Alternativ oder ergänzend könnte auch die jeweilige Laufzeitumgebung den Ausfall detektieren und entsprechende Maßnahmen ohne Verzögerung ergreifen. Diese sitzt jedoch auf demselben Host wie die Operatorinstanz und fällt beim Knotenabsturz typischerweise mit aus. Wie immer bei Ausfallsbetrachtungen könnte eine fehlerhaft programmierte Operatorinstanz, die immer bei der gleichen Selektion ausfällt, das System zum Stagnieren bringen. Dies ist z.B. durch Speicherzugriffsfehler wie Dereferenzierungen von Nullzeigern möglich. Das Modell kann somit nur von sporadisch auftretenden Fehlern schützen, nicht jedoch, wenn diese reproduzierbar sind.

Es ist auch möglich, dass stattdessen das erweiterte Crash-Recovery-Modell zum Einsatz kommt. Eine ausgefallene Operatorinstanz wird dann nicht durch eine neue ersetzt, sondern steht dem System nach der Regeneration wieder automatisch zur Verfügung. Dabei muss sie jedoch in einen definierten Anfangszustand versetzt werden, was durch die Laufzeitumgebung bereits ohnehin geschieht. Im Folgenden wird jedoch das allgemeinere Crash-Stop-Modell herangezogen.

3.4 Kommunikationsverbindungen

Die Operatoren und Operatorinstanzen kommunizieren über Kommunikationsverbindungen miteinander. Es werden in diesem Systemmodell vier wichtige Anforderungen und Eigenschaften vorausgesetzt, um die Problematik von Ausfällen in Kommunikationskanälen und Operatorinstanzen zu entkoppeln: [Kol12]

Keine Erzeugung: Wenn ein Operator ω_j oder ein Ereigniskonsument c ein Ereignis σ empfängt, so hat ein anderer Operator ω_i oder eine Ereignisquelle s dieses Ereignis σ gesendet. Der Kommunikationskanal generiert also von sich aus keine neuen Ereignisse.

Keine Vervielfältigung: Jedes von einem Operator ω_i oder einer Ereignisquelle s versendete Ereignis ω wird von einem anderen Operator ω_j oder eines Ereigniskonsumenten c höchstens einmal empfangen. Der Kommunikationskanal dupliziert also keine übertragenen Ereignisse.

Zuverlässige Auslieferung: Wenn ein Operator ω_i oder eine Ereignisquelle s ein Ereignis σ an einen anderen Operator ω_j oder einen Ereigniskonsumenten c sendet, der Absender nicht abstürzt und der Empfänger korrekt ist, wird schlussendlich das Ereignis σ auch empfangen. Ein Ereignis wird also letztlich korrekt übertragen, sofern die Kommunikationspartner keine Fehler im Gegenüber oder im Kanal vermuten.

Erhalt der Reihenfolge: Beim Senden von mehreren Ereignissen σ_i und σ_j durch einen Operator ω_i oder eine Ereignisquelle s werden diese in der gleichen Reihenfolge σ_i und σ_j am anderen Operator ω_j oder der Ereignissenke c empfangen. Der Kommunikationskanal wird also die gesendeten Ereignisse beim Übertragen nicht umsortieren.

3.5 Ereignisquellen

Das CEP-System soll Ereignisse verarbeiten, die aus der echten Welt stammen. Ereignisquellen – oder kurz nur Quellen genannt – sind dort als Komponenten angesiedelt und liefern diese Ereignisse ins CEP-System. Z.B. können das im einfachsten Fall Sensoren sein oder aber auch andere Computersysteme, die Signale verarbeiten oder weiterleiten. Informationen aus der Umgebung werden zu Ereignissen im Sinne des CEP-Systems verarbeitet, indem sie eventuell um Informationen für die dortige Verarbeitung angereichert werden. Auch müssen sie in ein für das CEP-System verwendbares Format umgewandelt werden. Somit

entsprechen Ereignisquellen den *Event Adapters* aus David Luckhams Buch *The Power of Events [...]* [Luc02].

Ereignisquellen müssen synchronisierte Systemuhren besitzen, um den Ereignissen synchronisierte Zeitstempel in Realzeit geben zu können. Diese Ereignisse sind die einfachsten im CEP-System und werden deshalb auch als *Basisereignisse* bezeichnet. Da Teile des Operators, insbesondere der Splitter als fehlerfrei anzusehen sind (vgl. Abschnitt 3.3.6 *Fehlermodell* auf Seite 29), benötigen die Ereignisquellen kein Protokoll oder Zwischenspeicher, um versendete Ereignisse für den Fehlerfall vorzuhalten, wie es im rein verteilten CEP-System der Fall wäre. Das Vorhalten der Ereignisse für die Wiederherstellung beim Ausfall einer Operatorinstanz übernimmt hier der Splitter.

3.6 Ereigniskonsumenten

Die Ereigniskonsumenten oder Ereignissenken sind die Stellen, in denen Ereignisse das CEP-System verlassen. Sie führen die komplexen Ereignisse den eigentlichen Interessenten zu. Diese müssen wie auch die Quellen ständig verfügbar sein, was z.B. durch eine redundante Auslegung geschehen kann. Alternativ könnte der letzte Operator in der Verarbeitungskette alle komplexen Ereignisse in seinem Sequenznummerngenerator vorhalten, um beim Wiederauftauchen im System die Ereignisse zu erhalten. Da jedoch in den allermeisten Fällen ohnehin eine zeitnahe Weiterverarbeitung der erkannten Situationen auf Interessentenseite im Vordergrund steht, verfolgt diese Diplomarbeit den zuerst genannten Ansatz.

3.7 Kontrollereignisse

Für das Serialisieren am Splitter und am Sequenznummerngenerator ist es nötig, so lange zu warten, bis sicher ist, dass kein Ereignis mehr eintrifft, das vor einem anderen in die Gesamteingangs- oder -ausgangsströme übernommen werden soll. Um hier die Wartezeiten zu reduzieren, sollten davor liegende Komponenten Kontrollereignisse senden, um zu signalisieren, dass kein solches Ereignis existiert. Dieses Problem kann zum Beispiel bei einer geringen Ereignissenderate auftreten. Mithilfe der empfangenen Kontrollereignisse lässt sich die Verarbeitungsgeschwindigkeit an den Splittern und Sequenznummerngeneratoren erhöhen.

3.8 Zeit und Asynchronität

Die Ereignisquellen sind die einzigen Komponenten, die hinreichend genau synchronisierte Uhren besitzen. Somit können sie erzeugte Ereignisse mit Zeitstempeln in Echtzeit ausstatten, die schließlich genutzt werden kann, um den Echtzeitbezug von eingetretenen Situationen beim Interessenten herzustellen.

Für Operatoren und deren Instanzen gibt es keine genauen Annahmen über zeitliche Abläufe in Bezug auf Verarbeitung und Kommunikation. Ein hohes Maß an Synchronität wird hierbei nicht benötigt. Allein für die Ausfallerkennung einer Operatorinstanz durch den Sequenznummerngenerator darf eine gewisse Zeit zwischen zwei von der Instanz gesendeten Ereignissen nicht überschritten werden.

3.9 Puffer mit FIFO-Eigenschaften

Da mehr Ereignisse eingehen als sofort verarbeitet werden können, stehen für alle Bestandteile eines Operators verschiedene Eingangspuffer zur Verfügung. Sowohl Splitter als auch Sequenznummerngenerator sowie Operatorinstanz mitsamt ihrer Laufzeitumgebung müssen eingegangene Ereignisse für die rasche, aber später erfolgende Weiterverarbeitung zwischenspeichern. Dies geschieht durch variable Puffer, auch Warteschlangen genannt. Dabei gilt die FIFO-Eigenschaft, d.h. zuerst eingereihte Ereignisse werden auch zuerst wieder verarbeitet.

So besitzt der Splitter beispielsweise für jeden Eingangsstrom des Operators einen eigenen Puffer. Über eine Eingangsleitung eintreffende Ereignisse gelangen in den jeweiligen Puffer. Sobald in jedem Eingangspuffer mindestens ein Ereignis eingangen ist, kann anhand der in Abschnitt 3.3.2 *Sequenzierung der Eingangsströme* beschriebenen Sortiervorschrift mindestens ein Ereignis aus einer der Puffer genommen und serialisiert werden.

Teilweise ist es jedoch auch nötig, Ereignisse in den Puffern zu priorisieren. So beispielsweise in der Laufzeitumgebung des Operators ω_i^j, wenn eine andere Operatorinstanz ω_i^k ausgefallen ist und daher die von ω_i^k zu bearbeitenden Selektionen an andere Instanzen – so auch an ω_i^j – neu verteilt werden sollen. Um rechtzeitig die Ereignisse der ausgefallenen Instanz zu verarbeiten, müssen die neu erhaltenen Selektionen eventuell bereits vor den schon existierenden Selektionen im Puffer verarbeitet werden. Dies leistet ein Prioritätspuffer, der nach Selektion sortiert die Ereignisse ausgibt. Dennoch bleibt die selektionsinterne Reihenfolge der Ereignisse gleich.

4 Parallelisierungsrahmenwerk

Das Parallelisierungsrahmenwerk ist die Grundlage, Ereignisströme konsistent aufzuspalten und so erst die parallelisierte Verarbeitung durch verschiedene Operatorinstanzen zuzulassen. Eine zugrundeliegende Idee dabei ist auch, dass bestehende, konventionelle Operatoren möglichst einfach in das System portiert werden können, ohne dass dabei mehr globales Wissen durch den Programmierer in die Operatoren eingebaut werden müsste. Der Operatorcode muss somit nur geringfügig modifiziert werden. Jedoch muss der Anwender das Wissen in den Splitter einfügen, wie Selektionen gebildet werden sollen. Die hierzu erforderlichen Informationen werden deshalb vom Parallelisierungsrahmenwerk eingespeist und auch verwaltet. Aufgabe dieser Diplomarbeit ist unter anderem das Implementieren eines solchen Rahmenwerks, da es bisher nur in der Theorie existiert. Es basiert grundlegend auf dem noch unveröffentlichten wissenschaftlichen Artikel *Parallelization of Continuous Complex Event Processing Operators with the ParSe middleware* von Ruben Mayer [May13b], dem Betreuer dieser Diplomarbeit. Die Wiederherstellung von ausgefallenen Operatorinstanzen entstammt hauptsächlich dem Artikel *Rollback-Recovery without Checkpoints in Distributed Event Processing Systems* [KMR+13]. Aus Zeitgründen – Hauptthema der Arbeit ist eigentlich die Zuverlässigkeit, nicht die Kodierung des zugrundeliegenden Systems – werden jedoch einige Vereinfachungen vorgenommen.

Das Rahmenwerk ist in Python 3 [Pyt13] implementiert und verwendet seinerseits Tornado 3.1.0 [Dar13], ein Webrahmenwerk und eine asynchrone Netzwerkbibliothek. Die Verwendung von Python vereinfacht die Implementierung, da es gute Abstrahierungsmöglichkeiten bietet. Tornado kommt hier allerdings nicht als Webserver sondern als Netzwerkbibliothek zum Einsatz.

4.1 Grundlegende Komponenten

In dem Rahmenwerk gibt es vier verschiedene Hauptkomponenten:

- Splitter
- Laufzeitumgebung für Operatorinstanzen
- Operatorinstanzen selbst
- Sequenznummerngenerator

Die Ereignisquellen und -senken sind außerhalb des CEP-Systems angesiedelt und sollten daher ursprünglich nicht als extra Komponenten im Rahmenwerk mitgeliefert werden. Denn sie können einfach bei Bedarf selbst erzeugt werden, etwa indem Sockets zum Versand bzw. Empfang genutzt werden. Für die Experimente stellte es sich jedoch als hilfreich heraus, diese auch im Rahmenwerk vorzuhalten. Somit sind auch rudimentäre Ereignisquellen und -konsumenten mit als Komponenten enthalten.

Die Topologie wird von Hand erstellt. Der Graph muss schrittweise von hinten nach vorne aufgebaut werden, da sich alle Komponenten beim Initialisieren automatisch zu ihren nächsten Nachbarn via TCP verbinden. Eine Ausnahme bildet der Rückkanal vom Sequenznummerngenerator zum Splitter, um fehlende Selektionen zurückzumelden. Diese Verbindung wird erst dann aufgebaut, wenn tatsächlich Wiederherstellungsereignisse gesendet werden sollen. Entsprechend muss jede Komponente alle ihre Nachfolger kennen. Für die Zukunft kann das System um verschiedene Algorithmen zur automatischen Konfiguration erweitert werden. Dies geht aber über diese Diplomarbeit hinaus.

Im Folgenden soll eine kurze Einführung in alle vier Hauptkomponenten und deren Abhängigkeiten gegeben werden. Nähere Informationen sind der mit Sphinx [Bra11] erzeugten englischsprachigen API-Dokumentation zu entnehmen. Im Quellcode selbst können aus den ebenfalls auf Englisch gehaltenen Kommentaren weiterführende Hinweise gewonnen werden. Beide Ressourcen sind auf der zu dieser Arbeit gehörenden CD enthalten und unter `doc/` bzw. `src/` zu finden.

Splitter Der Splitter führt verschiedene Eingangsereignisströme zusammen und teilt den serialisierten Gesamtstrom sofort wieder auf, um die angeschlossenen Laufzeitumgebungen mit den Selektionen zu versorgen. Er bekommt beim Instanziieren die zu verwendende Selektionsschnittstelle, ein Steuerprogramm zur Verteilung auf die Laufzeitumgebungen sowie die Adressen der zu bedienenden Laufzeitumgebungen übergeben. Die Selektionsschnittstelle muss vom Benutzer bereitgestellt werden; sie definiert die Selektionen auf den Ereignissen. Als Variante des Steuerprogramms ist im Lieferumfang des Rahmenwerks ein `RoundRobinScheduler` enthalten, der die erzeugten Selektionen abwechselnd auf eine der angeschlossenen Laufzeitumgebungen verteilt. In Abbildung 4.1 ist die Architektur des Splitters gezeigt. Im Gegensatz zu dem in [May13b] beschriebenen Modell gibt es in dieser Implementierung keine Aufgabenpakete (*jobs*). Diese Pakete dienen unter anderem dazu, mehrere Selektionen für eine Operatorinstanz im Voraus auf dem Splitter zu gruppieren, um so eine gleichmäßigere Auslastung der Operatorinstanzen zu erzielen, wenn die Selektionen unterschiedlich groß sind. Wenn sich zudem Selektionen stark überlappen, kann durch die Gruppierung von mehreren Selektionen in Aufgabenpakete der Netzwerkverkehr reduziert werden. Da aber Selektionen in dieser Arbeit immer gleich viele Ereignisse umfassen und nicht überlappend sind, entfällt diese Notwendigkeit.

Laufzeitumgebung Die Laufzeitumgebung dient dazu, dass der ursprüngliche Operatorcode ohne große Anpassung mit dem Parallelisierungsrahmenwerk zusammenarbeiten kann. Sie verwaltet die vom Splitter zugewiesenen Selektionen und gibt sie der Operatorinstanz zur

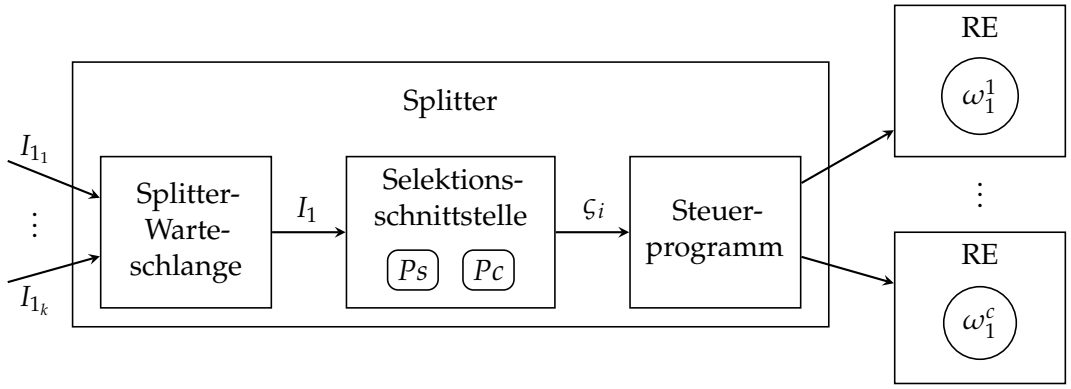


Abbildung 4.1: Die Architektur des Splitters. Der Benutzer stellt die Selektionsschnittstelle mit den beiden Prädikaten *Ps* sowie *Pc* bereit.

Verarbeitung so weiter, dass konsistente Ereignisströme erzeugt werden. Für die Operatorinstanz ist dabei jedoch nicht direkt sichtbar, dass sie von der Laufzeitumgebung ferngesteuert wird. Neben der Überwachung des Fortschritts leitet die Laufzeitumgebung die Ereignisse an den Sequenznummerngenerator weiter. In einer Laufzeitumgebung läuft immer genau eine Operatorinstanz, sofern diese nicht kurzzeitig beendet oder noch gar nicht gestartet wurde. Jede Laufzeitumgebung hat einen durch die Instanziierung fest zugeordneten Operatortypen zu verwalten. Im CEP-System kann es mehrere verschiedene Laufzeitumgebungen geben, die alle unterschiedliche Instanzen desselben Operatortypen ausführen. Eine Laufzeitumgebung stellt ihrer Operatorinstanz eine kleine Schnittstelle zur Verfügung, mit der sie Ereignisse empfangen und senden kann. Zu sendende Ereignisse werden von der Laufzeitumgebung immer mit einer neuen Sequenznummer versehen, die sich aus der aktuellen Selektion und einer fortlaufenden Nummer zusammensetzt (vgl. Absatz 4.2.1 *Sequenznummer* auf Seite 39). Fehlt diesem Ereignis der Start- und/oder Endzeitstempel, setzt die Laufzeitumgebung den fehlenden Zeitstempel gemäß dem Systemmodell auf den Startzeitpunkt des ersten Ereignisses in der Selektion. Wurde für die Selektion schon mindestens ein Ereignis produziert, so wird stattdessen der Zeitstempel des letzten erzeugten Ereignisses genutzt. Am nächsten Operator sortiert der Splitter anhand dieser Zeitinformation die eintreffenden Ereignisse; dadurch entsteht die Notwendigkeit, Ereignisse immer mit Zeitstempeln auszustatten. Gemäß Postels Devise „Be generous in what you accept, rigorous in what you emit“ zur Unix-Philosophie [Ray03], ist die Laufzeitumgebung für die Einhaltung der Zeitstempelregel verantwortlich. Dennoch sollte die Operatorinstanz die Zeitstempel der Ereignisse immer in streng monoton steigender Reihenfolge ausfüllen und sich an die in Abschnitt 3.2, Absatz *Zeitstempel* τ auf Seite 24 beschriebenen Hinweise halten. Außerdem bietet die Laufzeitumgebung der Operatorinstanz eine Möglichkeit, ihr mitzuteilen, wann diese ihre Korrelation auf einer Selektion beendet hat. Dies ist für die korrekte Funktionsweise der Sequenznummernvergabe sowie die Verwaltung der Selektionen durch die Laufzeitumgebung notwendig. Ereignisse verschachtelter oder direkt hintereinanderliegender Selektionen können performant in einem Durchgang abgearbeitet werden. Um falsche Ergebnisse bei nicht direkt aufeinanderfolgenden Selektionen zu verhindern, muss die Laufzeitumgebung den Operator jedoch bei Erhalt des Signals „Korrelation beendet“ stoppen und ihn für die nächste zu bearbeitende Selektion

neustarten. Bei der Initialisierung einer Operatorinstanz werden die Warteschlangen der Instanz mit den Ereignissen aus der gerade aktuellen Selektion gefüllt und der Instanz die Möglichkeit gegeben, benötigte Datenstrukturen vor Verarbeitungsbeginn aufzubauen.

Operatorinstanz Im Rahmenwerk ist eine Grundklasse für Operatoren bzw. Operatorinstanzen enthalten, die vom Benutzer implementiert werden muss. Eine optionale Methode zum Erledigen von Aufräumarbeiten ist das Gegenstück zur Startroutine der Operatorinstanz, in der alle Initialisierungen der benötigten Datenstrukturen vorgenommen werden können. Jede Operatorinstanz besitzt eine Warteschlange, in die die Laufzeitumgebung die eintreffenden Ereignisse der aktuellen Selektion einreiht. Die Operatorinstanz kann beliebig auf der Warteschlange arbeiten und Ereignisse auch mehrfach für eine Korrelation verwenden. Auf welcher genauen Selektion aktuell gearbeitet wird, ist dabei ausschließlich für die Laufzeitumgebung sichtbar. Allerdings muss sie die Laufzeitumgebung benachrichtigen, wenn sie die Korrelation abgeschlossen hat. Dies ist anhand des konkreten Anwendungsfalls und des Wissens darüber, wie Selektionen aufgebaut sind, möglich.

Sequenznummerngenerator Der Sequenznummerngenerator bringt die empfangenen Ereignisse in eine eindeutige Reihenfolge, indem er anhand der durch die Laufzeitumgebung vergebenen Sequenznummerntupel Selektionen erkennt und zu neuen, fortlaufenden, einfachen Skalaren umschreibt. So liegen die Sequenznummern in der darauffolgenden Komponente, dem Splitter oder der Senke, wieder wie gewohnt als natürliche Zahl vor. Ein inhaltlicher Vergleich von Ereignissen mit gleichen Sequenznummern, wie dies beim parallelen Verarbeiten derselben Selektion auftritt, erfolgt nicht. Sie werden als normale Duplikate behandelt und herausgefiltert. Der Sequenznummerngenerator erkennt Ausfälle von Operatorinstanzen, wenn diese ihm in einem bestimmten Zeitraum kein neues Ereignis überreicht haben. In dem Fall benachrichtigt er den Splitter, um die fehlende Selektion erneut verarbeiten zu lassen, sofern nicht eine andere Operatorinstanz dieselbe Selektion schon verarbeitet hat, was beim redundanten Verarbeiten auftritt. Somit kann der Splitter auch die Instanz als ausgefallen markieren.

4.2 Ereignisse

Die Ereignisse sind im Rahmenwerk als YAML-Objekte modelliert. YAML [BKEN09] – ursprünglich *Yet Another Markup Language*, jetzt *YAML Ain't Markup Language* – ist menschenlesbar komprimiertes JSON, *JavaScript Object Notation* [Cro13]. Über Listen, assoziative Felder sowie Skalare (Zeichenketten, Zahlen, Wahrheitswerte und dem Nullwert) lassen sich alle beliebigen Datenstrukturen darstellen. Dadurch, dass bei Strings in den meisten Fällen in YAML die umschließenden Anführungszeichen weggelassen werden können, lässt sich im Vergleich zu JSON etwas an Speicherplatz einsparen (vgl. Abbildung 4.2). Verglichen mit reinen Binärformatnachrichten oder Nachrichten mit einer festen Struktur, bei der die Nachrichtenfelder nicht explizit als Namen in der Nachricht selbst auftauchen, benötigt YAML für die Übertragung sehr viel Speicherplatz, bleibt jedoch deutlich unter vergleichbaren

```json
{
  "type":  "foo",
  "seq":   42,
  "start": 1383749574,
  "end":   1383749595,
  "data": {
    "sensors": [true, true,
                null, false]
  }
}
```

```yaml
{
  type:  foo,
  seq:   42,
  start: 1383749574,
  end:   1383749595,
  data: {
    sensors: [true, true,
              null, false]
  }
}
```

```xml
<?xml version="1.0"?>
<event type="foo"
       seq="42"
       start="1383749574"
       end="1383749595">
  <sensor value="true"/>
  <sensor value="true"/>
  <sensor value="null"/>
  <sensor value="false"/>
</event>
```

Abbildung 4.2: Beispieldarstellungen eines Ereignisses in verschiedenen Formaten kodiert: als JSON-Objekt (links), YAML-Objekt (Mitte) sowie möglichst kompake XML-Struktur (rechts).

```yaml
{type: temp, seq: 23, start: 495894, end: 496433, data: {temp: 21.34, loc: "GP"}}
```

Abbildung 4.3: Beispielereignis als einzeiliges YAML-Objekt serialisiert.

XML-kodierten Nachrichten. Der Vorteil von Nichtbinärformaten wie YAML ist, dass sich Ereignisse ziemlich einfach erweitern lassen oder nicht benötigte Informationen auch ganz fehlen können. Gerade dann, wenn nicht absehbar ist, welche zusätzlichen Informationen eventuell noch in Zukunft mit in die Ereignisse aufgenommen werden müssen, ist diese Flexibiliät sehr von Vorteil. Zudem erleichtern menschenlesbare Protokolle die Fehlersuche erheblich.

Die Ereignisse werden wie in Abbildung 4.3 als einzeilige YAML-Objekte serialisiert und mit einem Zeilenvorschub (`\n`) am Ende versehen. Somit kann der Parser auf der Empfangsseite sehr einfach auf einen abschließenden Zeilenvorschub prüfen und – korrektes Ereignissenden vorausgesetzt – sicher feststellen, dass die Ereignisinformation komplett ist. Soll in den Nutzdaten ein Zeilenvorschub Teil einer Zeichenkette sein, muss dieser, wie in YAML, Python und anderen Sprachen üblich, maskiert werden: `\n`. Abbildung 4.4 zeigt, dass ein YAML-Dokument auch hübsch formatiert in mehrere Zeilen aufgeteilt und in mehrere Abschnitte unterteilt werden kann, sodass aus einem Datenstrom problemlos auch mehrere Ereignisse geparst werden können. Allerdings ist die technische Umsetzung davon etwas komplizierter, außerdem bringt die hierzu nötige Einrückung von mehrzeiligen Abschnitten auch größere Datenmengen mit sich, die übertragen werden müssen. Wegen der Einfachheit und Kompaktheit entscheide ich mich für einzeilige Nachrichten mit abschließendem Zeilenvorschub.

Wie es teilweise aus den Beispielen schon ersichtlich ist, enthalten Ereignisse die folgenden Felder. Hier soll nur ein grober Überblick gegeben werden, genaue Informationen sind entweder der API-Dokumentation oder dem Quellcode selbst zu entnehmen.

`type`: Der Ereignistyp, vgl. Abschnitt 3.2, Absatz *Ereignistyp*. Einige Ereignistypen sind für das CEP-System reserviert und haben die folgenden Bedeutungen:

```yaml
---
type: foo
seq: 42
start: 1383749574
end: 1383749595
data:
  sensors:
    - true
    - true
    - null
    - false
---
type: bar
seq: 43
start: 1383749597
end: 1383749600
data:
  warning: true
  level: 3
  since: 2013-11-04T23:12:54Z
  message: "We need help with \"eggs and spam\""
---
...
```

Abbildung 4.4: Strom von Ereignissen als gut menschenlesbares, formatiertes YAML-Dokument.

`hello`: Mit Begrüßungsereignissen stellen sich Komponenten ihren Nachfolgern beim Verbindungsaufbau zuerst vor. Sie enthalten im Ereignisfeld `procname` den global eindeutigen Prozessnamen des Absenders. Diese Information wird z.B. für die eindeutige Sortierung im Splitter verwendet.

`pulse`: Pulsereignisse dienen zur effizienteren Verarbeitung an nachfolgenden Komponenten. Nähere Information zu Pulsereignissen sind im folgenden Unterabschnitt gegeben.

`recover`: Mit Wiederherstellungsereignissen benachrichtigt der Sequenznummerngenerator den Splitter über ausgefallene Operatorinstanzen.

`ack`: Mit Bestätigungsereignissen quittieren Empfänger die erfolgreiche Auslieferung an die Konsumenten. Siehe auch Abschnitt 4.3 *Verbindungen*.

`start`: Wie im Abschnitt 3.2 im Absatz *Ereigniszeitstempel* τ erläutert ist, können Zeitstempel nicht nur Zeitpunkte sondern auch Zeitspannen sein. Die Ereignisse haben hierfür zwei Felder, `start` enthält den Beginn des Ereignisses.

`end`: Das zeitliche Ereignisende. Soll ein Zeitpunkt und keine Zeitspanne beschrieben werden, wird dieses Feld auf den gleichen Wert wie `start` gesetzt.

`seq`: Die Sequenznummer des Ereignisses.

`selections`: Liste aller Selektionen, in denen dieses Ereignis enthalten ist. Die aktuelle Implementierung kann jedoch momentan nicht an allen Stellen mit überlappenden Selektionen umgehen, was nach dem eingangs beschriebenen Systemmodell (vgl. Abschnitt 3.3.3 *Selektionsbildung* auf Seite 27) auch nicht gefordert ist.

`data`: Die Nutzdaten des Ereignisses.

4.2.1 Pulsereignisse

Pulsereignisse werden von allen Komponenten erzeugt, wenn sie keine weiteren Ereignisse mit eigentlicher Information mehr verschicken können. Dies tritt z.B. dann ein, wenn eine Quelle nur mit einer recht geringen Ereignisrate sendet. Gäbe es keine Pulsereignisse, müssten die Weichen im System (also Splitter und Sequenznummerngeneratoren) unnötig lange warten, um mit der Serialisierung der Ereignisse aus anderen Kanälen mit zeitlich häufiger eintreffenden Ereignissen fortfahren zu können. Das Rahmenwerk unterscheidet zwei verschiedene Arten von Pulsereignissen, die sich auf das Sortierkriterium an der nächsten Komponente beziehen:

1. Pulsereignisse mit dem Startzeitstempel als Sortierkriterium
2. Pulsereignisse mit der Sequenznummer als Sortierkriterium

Startzeitstempel Dies ist die am häufigsten auftretende Form von Pulsereignissen. Hierbei sortiert die nachfolgende Stelle im System alle Ereignisse inklusive dem Pulsereignis aller eingehenden Ereignisströme und beachtet nur den Startzeitstempel der Ereignisse. Bei mehrfach vorkommenden gleichen Startzeitstempeln entscheidet der Prozessname deterministisch über die Reihenfolge der fraglichen Ereignisse. Dies führt zu zeitlich geordneten Ereignissen in den Ausgangskanälen. Ereignisquellen und Splitter verwenden diese Art von Pulsereignissen.

In den Laufzeitumgebungen und den darin ausgeführten Operatorinstanzen selbst müssen keinerlei Datenströme miteinander verschmolzen werden. Daher ist in diesen keine topologische Sortierung nötig. Die Laufzeitumgebung erzeugt von sich aus keine Pulsereignisse, die sie dem Operator weiterreichen würde. Allerdings kann natürlich der Operator Pulsereignisse erzeugen und über die Laufzeitumgebung an den Sequenznummerngenerator weiterreichen lassen. Hierbei wird es sich höchstwahrscheinlich auch um startzeitstempelgesteuerte Pulsereignisse handeln.

Sequenznummer Meldet die Operatorinstanz das Ende eines Korrelationsschrittes – also die erfolgreiche Abarbeitung einer Selektion – an die Laufzeitumgebung zurück, so generiert diese für den nachfolgenden Sequenznummerngenerator ein Pulsereignis. Das Pulsereignis ist dazu da, dem Sequenznummerngenerator dieselbe Information mitzuteilen, nämlich dass die Selektion beendet ist und daher keine Ereignisse mehr eintreffen können, die zu dieser Selektion gehören. Somit kann dieser das Serialisieren der übrigen Selektionen von anderen Operatorinstanzen anstoßen. In diesem Pulsereignis ist jedoch nicht der Zeitstempel relevant,

sondern die Sequenznummer, die sich in dem Fall als Tupel $(\textit{beendete Selektion} + 1, -1)$ ausdrückt. Da -1 in der zweiten Tupelkomponente niemals sonst auftritt (0 wäre die erste gültige Nummer einer Selektion), führt dies zu keinen Komplikationen.

4.3 Verbindungen

Als Kommunikationskanäle kommen zwischen den einzelnen Komponenten im CEP-System TCP-Verbindungen zum Einsatz. Damit ist gewährleistet, dass sich Nachrichten im Datenstrom vom selben Absender zum gleichen Empfänger nicht überholen oder verloren gehen. Im Systemmodell ist im Abschnitt 3.4 *Kommunikationsverbindungen* auf Seite 30 ausgeschlossen, dass es zu Fehlern kommen kann, die den Kommunikationskanal betreffen.

Dennoch sind bereits rudimentäre Einrichtungen vorhanden, um Verbindungsfehler abzufedern. Bricht eine Verbindung zusammen, kann die Empfängerseite mithilfe von Wiederherstellungsereignissen versuchen, die dadurch zwischenzeitlich verlorengegangenen Ereignisse von der Gegenseite durch erneutes Senden wiederherzustellen. Zu Beginn der Diplomarbeit war vorgesehen, dass die Senderseite nach einem erneuten Verbindungsaufbau alle angeforderten Ereignisse ein weiteres Mal ausliefern kann. Werden Ereignisse versendet, so wird auf Senderseite ein Puffer befüllt, aus dem die Wiederherstellung erfolgen kann. Auch wenn eine CEP-Komponente abgestürzt und schließlich wieder verfügbar ist, können auf diese Weise verpasste Ereignisse empfangen werden. Dies hat zur Folge, dass auch die Ereignisquellen über diesen Mechanismus verfügen müssen. Damit die Ereignissendepuffer nicht überlaufen, muss die Senke beim Erhalt eines Ereignisses selbiges durch ein Bestätigungsereignis quittieren. Jede Komponente löscht bei Erhalt eines Bestätigungsereignisses alle bereits gesendeten Ereignisse bis inklusive dem aktuell Bestätigten aus ihrem Sendepuffer. Komponenten, die Sequenznummern übersetzen, halten daher eine Abbildung von umgeschriebenen zu ursprünglichen Sequenznummern vor, um so ihre Vorgänger mit korrekt sequenzierten Bestätigungsereignissen zu versorgen. Da all dies direkt in der Verbindung implementiert ist, ist dies für die Anwendung transparent. Aufgrund des Umstiegs auf das einfachere Systemmodell sind aber schon entsprechend vorhandene Codestellen im Bezug auf das Reparieren von Kommunikationsfehlern nun auskommentiert.

4.4 Warteschlangen

Um empfangene Ereignisse vorzuhalten, kommen in den Komponenten verschiedene Warteschlangentypen zum Einsatz. Dort verweilen die Ereignisse, bis sie zum Verarbeiten herausgenommen werden. Jede Komponente hat leicht andere, spezielle Anforderungen an die Serialisierung der Ereignisse, weshalb es im Rahmenwerk vier Typen von Warteschlangen gibt.

Splitter-Warteschlange Eine Aufgabe des Splitters ist die Verteilung der auf dem Gesamteingangsstrom I_i gebildeten Selektionen an die Operatorinstanzen. Zuvor muss jedoch erst I_i erzeugt werden. Dies geschieht in der Splitter-Warteschlange, die man als *zusammenführende FIFO-Warteschlange* bezeichnen könnte. Die Erzeugung von I_i geschieht beim Ausreihen des nächsten Elements aus dieser Warteschlange. Die Splitter-Warteschlange enthält ihrerseits für jeden Vorgänger eine eigene FIFO-Eingangswarteschlange, in der empfangene Ereignisse eingereiht werden. Alle Eingangswarteschlangen müssen mindestens ein Ereignis enthalten. Ist mindestens eine leer, muss der Ausreihversuch erfolglos abgewiesen werden. Denn es könnte sein, dass aufgrund hoher Netzwerklast das eigentlich nächste zu serialisierende Ereignis noch nicht in der momentan leeren Eingangswarteschlange angekommen ist. Sind alle Eingangswarteschlangen mit mindestens einem Ereignis gefüllt, können ein oder mehrere Ereignisse aus der Splitter-Warteschlange herausgereicht werden. Hierzu werden alle ersten Ereignisse aus den Eingangswarteschlangen betrachtet und das älteste, also das mit dem kleinsten Startzeitstempel, serialisiert. Sind Ereignisse gleich alt, werden die Prozessnamen der in Frage kommenden Vorgänger alphabetisch verglichen. Dieses Verfahren wird so lange wiederholt, bis mindestens eine Eingangswarteschlange leer ist. Bei der Splitter-Warteschlange kann es also trotz mit Ereignissen gefüllten Eingangswarteschlangen dazu kommen, dass kein Ereignis herausgenommen werden kann.

Sequenznummerngenerator-Warteschlange Diese dient dazu, den Ausgangsstrom O_i zu erzeugen. Auch sie enthält für alle angeschlossenen Vorgänger, also Operatorinstanzen bzw. Laufzeitumgebungen eine eigene Eingangswarteschlange, in die die erzeugten Ereignisse eingereiht werden. Für alle Eingangswarteschlangen verwaltet sie die zwei erwarteten Sequenznummern, die entweder „nächstes Ereignis in der Selektion“ oder „Ende der Selektion“ kodieren. Erst wenn eine Selektion komplett ausgereiht wurde, kann auf die Darauffolgende gewechselt werden. Auch bei der Sequenznummerngenerator-Warteschlange kann es vorkommen, dass trotz enthaltenen Ereignissen ein Ausreihen nicht möglich ist. Dies ist dann der Fall, wenn für eine Selektion noch weitere Ereignisse ausstehen. Sie ist ebenso eine Art *zusammenführende FIFO-Warteschlange*, dennoch arbeitet sie im Detail anders als die Splitter-Warteschlange. Letztere arbeitet eher gleichmäßig die Eingangswarteschlangen ab, während Erstere diese eher blockmäßig verarbeitet.

Laufzeitumgebungs-Warteschlange In den Laufzeitumgebungen kommt statt dem FIFO-Prinzip eine *Prioritätswarteschlange* zum Einsatz. Hier müssen zwar keine Ströme miteinander verschmolzen werden, aber es kann vorkommen, dass Ereignisse andere überholen. Sei ω_i^j eine Operatorinstanz, in deren Laufzeitumgebung die Selektionen ς_4 und ς_9 in der Laufzeitumgebungs-Warteschlange enthalten sind, wobei ω_i^j gerade auf ς_4 arbeitet. Stürzt eine andere Operatorinstanz ω_i^k ab, müssen alle ihr zugeteilten Selektionen, etwa ς_3 und ς_8 an die übrigen Operatorinstanzen neu vergeben werden. ω_i^j bekommt vom Splitter z.B. ς_3 zugeteilt. Um die Verzögerung möglichst gering zu halten, muss die Laufzeitumgebung

von ω_i^j nun ς_3 nach vorne[1] in der Laufzeitumgebungs-Warteschlange schieben. Es wird also nach Selektion aufsteigend priorisiert. Die Ereignisreihenfolge innerhalb einer Selektion bleibt hiervon jedoch unberührt. ω_i^j kann nun die Selektion ς_4 vollständig abarbeiten und bekommt von ihrer Laufzeitumgebung dann ς_3 nach einem Neuaufsetzen zur Verarbeitung überreicht. Die Operatorinstanz ω_i^j muss deshalb beendet und neu eingerichtet werden, da die neue Selektion ς_3 nicht direkt auf ς_4 folgt. Technisch wird in der Laufzeitumgebungs-Warteschlange jeder Selektion eine separate Warteschlange zugewiesen, um so Ereignisse verschiedener Selektionen nicht zu vermischen. Diese Gefahr bestünde gerade dann, wenn eine zeitlich spätere Selektion ς_4 aktiv ist und eine frühere Selektion ς_3 als nächstes verarbeitet werden soll. Wie auch bei den beiden erstgenannten Warteschlangen im Splitter und Sequenznummerngenerator kann es hier ebenfalls dazu kommen, dass die Laufzeitumgebungs-Warteschlange Ereignisse enthält, die aber nicht herausgenommen werden dürfen, da eine Selektion noch aktiv ist und auf weitere Ereignisse wartet.

Operatorinstanz-Warteschlange Die einfachste Warteschlange ist die Operatorinstanz-Warteschlange, die eine einfache *FIFO-Warteschlange* ist. Von der Laufzeitumgebung werden die Ereignisse der akuellen Selektion in diese Warteschlange eingereiht. Die Instanz kann diese sich zum Verarbeiten herausnehmen. Hier kann es im Gegensatz zu den anderen drei aufgeführten Warteschlangen nicht dazu kommen, dass trotz gefüllter Warteschlange keine Ereignisse entnommen werden dürfen.

4.5 Wiederherstellung von Operatorinstanzen

Die Wiederherstellung im Fehlerfall funktioniert ähnlich wie in [KMR+13] beschrieben. Statt eines separaten, perfekten Fehlerdetektors, der Knotenausfälle ohne Verzögerung erkennen kann, kommt in dieser Implementierung der Sequenznummerngenerator für die Aufgabe zum Einsatz. Er stellt anhand von ausbleibenden Ereignissen innerhalb einer Zeitspanne den Ausfall einer Operatorinstanz fest. Diese Zeit kann in Experimenten so gewählt werden, dass keine falsch-positiven Ausfälle erkannt werden, also dass keine Operatorinstanzen als abgestürzt markiert werden, obwohl sie in Wirklichkeit fehlerfrei arbeiten. Allerdings ist dieser Ansatz nicht verzögerungsfrei.

Bei einem erkannten Ausfall benachrichtigt der Sequenznummerngenerator den Splitter über die fehlende Selektion. Dies geschieht über einen Rückkanal, der bei der ersten Verwendung einmal aufgebaut wird. Der Splitter kann in seiner internen Verwaltung anhand der gemeldeten Selektion die ausgefallene Instanz ermitteln und diese als offline markieren, um ihr in Zukunft keine weiteren Selektionen zur Verarbeitung zuzuteilen. Die vom Sequenznummerngenerator als fehlend gemeldete Selektion kann vom Splitter je nach ursprünglicher Verteilung erneut in Auftrag gegeben werden. Hat die ausgefallene Operatorinstanz zuvor weitere Selektionen vom Splitter erhalten, werden diese ebenfalls auf neue Instanzen verteilt –

[1] wenn vorne an einer Warteschlange die Elemente entnommen und hinten eingereiht werden

zumindest dann, wenn sie nur an die eine ausgefallene Instanz verteilt wurden. Wurde eine Selektion redundant zwei Operatorinstanzen zugeteilt, so muss vom Splitter nichts weiter veranlasst werden.

Die Laufzeitumgebungen, die die wiederherzustellenden Selektionen erhalten, bevorzugen dabei diese neuen, zeitlich früheren Selektionen in ihren Wartepuffern. Nachdem eine Operatorinstanz die aktuelle Selektion fertig verarbeitet hat, werden ihr von der Laufzeitumgebung eventuell zuerst die zu rettenden Selektionen übergeben, sodass die Selektionen aufsteigend sortiert verarbeitet werden können. Die Einhaltung der Reihenfolge ist wichtig, um die Ergebnisse am Ende rechtzeitig auszuliefern. Der Sequenznummerngenerator kann bis zum Erhalt der fehlenden Selektion keine weiteren, schon bereitstehenden Ereignisse anderer Selektionen serialisieren. Dies würde sonst zu einem inkonsistenten Ausgangsstrom und damit zu falschen komplexen Ereignissen führen.

Nach einer Minute steht die ausgefallene Operatorinstanz dem System in der aktuellen Implementierung automatisch zur Verfügung. Sie wird dann vom Splitter wieder mit neuen Selektionen versorgt.

5 Zielstellung

5.1 Motivation

Rechtzeitigkeit hängt einerseits von der Zeit ab, die eine Komponente für die Verarbeitung benötigt, andererseits auch vom Füllstand ihrer Warteschlange. Splitter und Sequenznummerngenerator haben eine recht einfache und schnell zu erledigende Aufgabe zu erfüllen. Sie puffern die Ereignisse nur solange, bis diese serialisiert werden können, was in der Regel rasch geschieht. Demnach ist die Gefahr des Warteschlangenüberlaufs und einer langen Verarbeitungszeit an diesen beiden Komponenten äußerst gering.

Mit Raten ist im Folgenden immer die Anzahl der Selektionen pro Zeiteinheit gemeint. Die eigentliche Verarbeitung der Ereignisse läuft in den Operatorinstanzen ab und kann sehr aufwändig werden. Verarbeitungsraten der Operatorinstanzen lassen sich nur sehr schwer bis gar nicht optimieren, daher muss der zweite Parameter der Rechtzeitigkeit – die Warteschlangenauslastung der Laufzeitumgebung – betrachtet werden. Diese ist über den Parallelisierungsgrad regelbar. Mit dem Parallelisierungsgrad kann damit letztlich auch das zeitliche Verhalten der Operatorinstanzen beeinflusst werden. Alle anderen Verzögerungen, wie Eingangs- oder Verarbeitungsgeschwindigkeiten, sind vom Parallelisierungsgrad unabhängig. Zur Untersuchung des Parallelisierungsgrades können also die Warteschlangenauslastungen analysiert werden. Demnach muss unbedingt eine gegebene Maximalgröße des Puffers eingehalten werden, um auch beim Ausfall einer Operatorinstanz einerseits weiterhin Rechtzeitigkeit garantieren zu können und andererseits den Durchsatz nicht einbrechen zu lassen.

5.2 Zielstellung

Untersuchungsgegenstand dieser Diplomarbeit ist die Größe des Parallelisierungsgrads, damit bei schwankenden Lasten und Verarbeitungszeiten mit probabilistischer Verteilung eine maximale Warteschlangenauslastung in den Laufzeitumgebungen eingehalten wird. Dabei können Fehler auftreten, die eine Operatorinstanz für eine Zeit lang zum Ausfall bringen, schlussendlich wird sie oder eine Ersatzinstanz jedoch wieder im System mitarbeiten. Für die Wiederherstellung gibt es zwei grundsätzlich verschiedene Verfahren:

Active Standby: hierbei werden Selektionen immer redundant verarbeitet [VKR11].

Savepoint-Recovery: hierbei werden Selektionen nur bei einem Ausfall erneut verteilt. Im Gegensatz zu dem in [KMR+13] geschilderten Verfahren sollen Selektionen in dieser

Arbeit jedoch sofort neu an die schon bereitstehenden, anderen Instanzen vergeben werden.

Im Gegensatz zu Active Standby müssen im Fall von Savepoint-Recovery verteilte Selektionen neu vergeben werden und es steht bis zur Wiederherstellung eine Instanz weniger zur Verfügung. Diese Auswirkungen müssen bei der Berechnung des Parallelisierungsgrades mit einbezogen werden.

Abhängig vom verwendeten Verfahren zur Wiederherstellung soll ein optimaler Parallelisierungsgrad ermittelt werden, zudem sollen die Verfahren miteinander verglichen werden. Rechtzeitigkeit und 100-prozentige Korrektheit müssen sowohl im Normal- als auch im Fehlerfall gewährleistet sein. Da die Anpassung des Parallelisierungsgrades Zeit braucht, ist eine Überprovisionierung erforderlich. Diese soll aber möglichst klein sein, um dennoch ein effizientes System zu erhalten. Die einzuhaltende, optimale Pufferauslastung wird von außen vorgegeben und kann theoretisch von einem Algorithmus im Voraus berechnet werden, der jedoch nicht Gegenstand dieser Diplomarbeit ist. In mehreren Experimenten soll schließlich das Modell überprüft und validiert werden.

5.3 Anforderungen an das CEP-System

Um die oben genannten Ziele zu erreichen, müssen einige Anforderungen an das CEP-System gestellt werden. Diese werden im Folgenden kurz beschrieben.

5.3.1 Hohe Skalierbarkeit

CEP-Systeme verarbeiten typischerweise große Datenmengen in Echtzeit. Daher muss das System gut skalierbar sein, d.h. bei höherer Last einen entsprechenden höheren Durchsatz liefern. Dies ist aber durch die Wahl eines geeigneten Parallelisierungsgrads einfach lösbar. Hinsichtlich der Untersuchungen muss hierbei nichts weiter beachtet werden, da die Eingangsrate am Splitter probabilistisch verteilt ist.

5.3.2 Echtzeit und geringe Verzögerungszeit

CEP-Systeme dienen zur sofortigen Analyse von Ereignissen in Echtzeit. Die schnelle Erkennung von Situationen ist maßgeblich für die zeitnahe Ergreifung von Maßnahmen durch die Ereigniskonsumenten. Wird eine Situation zu spät erkannt, ist sie wertlos oder kann unmittelbare, negative Konsequenzen für den Kosumenten nach sich ziehen (vgl. auch Abschnitt 2.4.2 *Echtzeit/Rechtzeitigkeit*). Daher ist die Einhaltung einer möglichst geringen Verzögerung unbedingt erforderlich. Wie eingangs erläutert, soll dies anhand einer garantierten Maximalpufferauslastung erfolgen. Die Rechtzeitigkeit muss auch im Fehlerfall gewährt sein.

5.3.3 Hohe Korrektheit der Ergebnisse

Wie in Abschnitt 2.4.1 *Korrektheit* beschrieben, werden an das CEP-System hier harte Korrektheitsanforderungen gestellt. So müssen die erkannten Situationen tatsächlich auch eingetreten und durch die Ereignisquellen berichtet worden sein. Beim Ausfall einer Operatorinstanz dürfen keine komplexen Ereignisse auf dem Weg zum Konsumenten verloren gehen und auch keine Zusätzlichen erzeugt werden. Falsch-positive und falsch-negative Ereignisse müssen also vollständig ausgeschlossen werden.

5.3.4 Hohe Ausfallsicherheit

Die Verfügbarkeit von CEP-Systemen soll generell sehr hoch sein, auch wenn Ausfälle auftreten. Nach außen hin darf im Fehlerfall keine Verhaltensabweichung erkennbar sein, d.h. Ereignisse dürfen nicht verloren gehen oder modifiziert werden. Zudem dürfen die zugesicherten Rechtzeitigkeitsgarantien nicht verletzt werden. Splitter, Sequenznummerngenerator sowie Laufzeitumgebungen und Kommunikationskanäle sind als zuverlässig zu betrachten und werden in dieser Arbeit nicht untersucht. Als Zuverlässigkeitsanforderung ist in dieser Arbeit in den Laufzeitumgebungen der durchschnittliche Pufferfüllgrad $Q(t)$ einzuhalten, der in ganzen Selektionen gemessen wird.

5.3.5 Hohe Effizienz

Um die Korrektheit der Ereignisse und die Rechtzeitigkeit zu garantieren, müssen bezüglich der Effizienz einige Abstriche in Kauf genommen werden (vgl. Abschnitt 2.4.3 *Systemleistung/Effizienz*). Dies geschieht beispielsweise durch redundante Auslegung und geringere Auslastung der Operatorinstanzen, um im Fehlerfall trotzdem schnell die korrekten komplexen Ereignisse auszuliefern. Dennoch sollen möglichst wenig Ressourcen verwendet werden, um die anderen Anforderungen umzusetzen.

6 Problemlösung

6.1 Warteschlangentheorie

Um Rechtzeitigkeit garantieren zu können, ist es nötig, die Warteschlangen der Operatorinstanzen bzw. Laufzeitumgebungen näher zu betrachten. Diese werden mit Ereignissen vom Splitter nach und nach befüllt sowie während der Verarbeitung sukzessive geleert. Daher spielt die Verzögerung durch diese Puffer im Gesamtsystem eine große Rolle. Pufferauslastungen können mithilfe der Warteschlangentheorie mathematisch berechnet werden. Diese liefert eine pessimistische Abschätzung der Füllgrade und bietet sich an, da die Eingangs- und Verarbeitungsraten als Wahrscheinlichkeitsverteilungen vorliegen. Die Verteilungen können beispielsweise von schwankenden Verarbeitungszeiten durch unterschiedliche Selektionsgrößen oder schwankenden Lasten am Vorgänger herrühren. Wir wollen dieses Modell zur Analyse des notwendigen Parallelisierungsgrads nutzen, um Pufferauslastungen in den Laufzeitumgebungen mit einer vorgegebenen Wahrscheinlichkeit unter einem ebenfalls vorgegeben Grenzwert zu halten. Dieser Abschnitt gibt eine kurze Einführung in die wichtigsten Punkte zu Warteschlangen.

Abbildung 6.1 zeigt den groben Aufbau eines Ausschnitts aus einem verteilten und parallelisierten CEP-System. Verschiedene Vorgänger oder Ereignisquellen s führen dem Splitter Ereignisse mit der Gesamteingangsrate λ Ereignisse pro Zeiteinheit zu, welche poissonverteilt ist. Ein Poissonprozess wird z.B. dann erreicht, wenn die Quellen Ereignisse mit einer Exponentialverteilung erzeugen und versenden.

Der gezeigte Operator ω_1 hat den Parallelisierungsgrad $c = 3$, d.h. es gibt drei Operatorinstanzen ω_1^1, ω_1^2 und ω_1^3, die vom Splitter gleichmäßig mit Ereignissen gespeist werden. Jede Operatorinstanz erhält somit nach dem Ringverteilungsverfahren eine Ereigniseingangsrate $\mu = \frac{\lambda}{c}$ und verarbeitet die gepufferten Ereignisse aus der Warteschlange q_i mit einer Rate ν. Komplexe Ereignisse verlassen schließlich den Operator über den Sequenznummerngenerator und erreichen die Nachfolgeoperatoren oder Ereigniskonsumenten k.

6.1.1 Warteschlangentypen

In der Warteschlangentheorie gibt es eine Vielzahl unterschiedlicher Warteschlangentypen. Anhand der als Standard etablierten Kendall-Notation [Ken53] können diese detailliert beschrieben werden. Hierzu wird folgendes Schema verwendet: $A/S/c$ bzw. $A/S/c/K/D$. Von grundlegender Bedeutung sind bei der Typbestimmung die folgenden drei Kriterien [Bos]:

Ankunftsprozess A: die statistische Verteilung neuer Einreihungen in die Warteschlange.

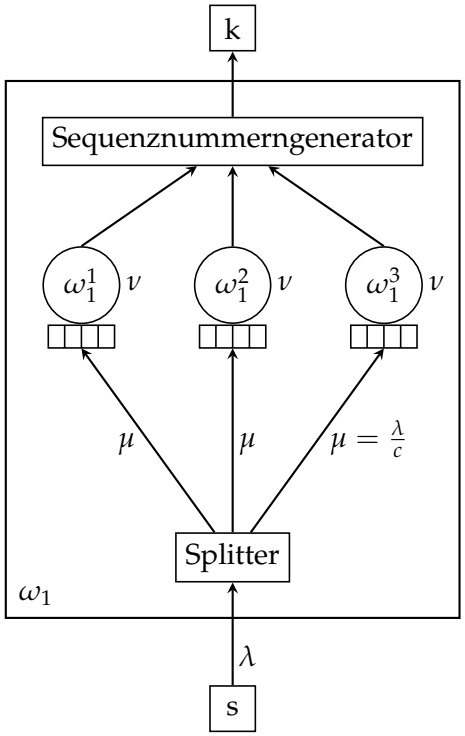


Abbildung 6.1: Verschiedene Ströme an einem Operator ω_1 aus einem parallelisierten und verteilten CEP-System [May13a].

Serviceprozess S: die statistische Verteilung von Ausreihungen aus der Warteschlange. Für die Kodierung von Ankunfts- und Serviceprozess werden verschiedene Buchstaben verwendet, z.B. steht M für eine Exponentialverteilung oder D für eine konstante Verteilung. Daneben gibt es noch eine große Auswahl weiterer Verteilungen, die hier aber mangels Relevanz nicht diskutiert werden. Die absolute Eingangsrate des Ankunftsprozesses darf nicht größer sein als die des Serviceprozesses multipliziert mit der Serviceeinheitanzahl, da sonst die Anzahl der belegten Plätze in der Warteschlange stetig anwächst.

Serviceeinheitanzahl c: die Anzahl der Prozesse, die Elemente aus der Warteschlange nehmen. Hierbei müssen die einzelnen Serviceprozesse bezüglich ihrer Aufgabenstellung immer identisch sein. Die Anzahl muss ≥ 1 sein.

Daneben gibt es noch zwei weitere Punkte, die ebenfalls bestimmte Eigenschaften einer Warteschlange implizieren:

Kapazität K: die Größe der Warteschlange und damit Anzahl an vorhandenen Wartepuffern. Wird diese Größe nicht explizit angegeben, so beträgt $K = \infty$, d.h. die Warteschlange kann beliebig viele Elemente aufnehmen und kann nicht überlaufen. Hat die Warteschlange eine endliche Kapazität, wird ein Einreihen in eine volle Warteschlange abgewiesen.

Abfertigungsdisziplin D: nach welcher Strategie Elemente in die Warteschlange ein- bzw. aus ihr ausgereiht werden. Dies kann etwa FIFO (First In First Out), LIFO (Last In First Out) oder auch eine völlig zufällige Ausreihung sein. Sofern nicht anders gekennzeichnet arbeiten Warteschlangen standardmäßig nach dem FIFO-Prinzip.

6.1.2 M/M/1-Warteschlangen

Für unsere weiteren Betrachtungen legen wir M/M/1-Warteschlangen zugrunde. Das heißt, die Verteilungen der hineingelegten und herausgenommenen Elemente der Warteschlangen sind jeweils poissonverteilt. Zudem gibt es einen einzelnen Serviceprozess, der Elemente aus der Warteschlange nimmt. Die Warteschlange ist unendlich groß, d.h. sie kann niemals volllaufen und daher auch keine Einreihungen abweisen. Genau genommen liegen auf den Laufzeitumgebungen zwar Prioritäts- und keine FIFO-Warteschlangen vor, in diesem speziellen Fall macht das jedoch für die Betrachtung keinen Unterschied.

Nach [May13a] und [GS01] lassen sich folgende Eigenschaften feststellen. Sei $Q(t)$ die Anzahl der Elemente in der Warteschlange zum Zeitpunkt t, also der Pufferfüllgrad. Die Auslastung ρ einer Operatorinstanz, wie in Abbildung 6.1 auf Seite 50 gezeigt, lässt sich aus dem Verhältnis des eingehenden Teilstroms λ zum ausgehenden Teilstrom ν folgendermaßen bestimmen:

$$\text{Auslastung } \rho = \frac{\text{Eingangsrate } \mu}{\text{Verarbeitungsrate } \nu}$$

Die Auslastung ρ muss dabei < 1 sein, da sich bei $\rho \geq 1$ der Pufferfüllgrad nie stabilisieren kann und bei $\rho > 1$ die Warteschlange auf jeden Fall stetig anwächst, d.h. die Operatorinstanz maßlos überlastet ist. Die Wahrscheinlichkeit, dass der Pufferfüllgrad $Q(t)$ zur Zeit t n beträgt, sich also n Elemente in der Warteschlange befinden, ist:

$$P(Q(t) = n) = (1 - \rho) \cdot \rho^n$$

Die Wahrscheinlichkeit, dass der Pufferfüllgrad $\leq n$ ist, ist gleich der Summe über alle Teilwahrscheinlichkeiten von einer leeren Warteschlange bis hin zu einer mit genau n Elementen:

$$P(Q(t) \leq n) = \sum_{i=0}^{n} (1 - \rho) \cdot \rho^i$$

Anhanddessen kann der Parallelisierungsgrad c bestimmt werden, der erforderlich ist, um die einzelnen Operatorinstanzen im Normalbetrieb nicht zu überlasten und den geforderten Warteschlangenfüllgrad $Q(t)$ mit der spezifizierten Wahrscheinlichkeit p_{soll} einhalten zu können. Um jedoch auch im Fehlerfall, d.h. bei Ausfall einer Operatorinstanz, diese Garantie einzuhalten, muss c neu bestimmt werden. Durch den Operatorinstanzausfall müssen die dieser Instanz zugewiesenen Ereignisse neu an die übrigen $c - 1$ Operatorinstanzen verteilt werden. Als Annahme liegt zugrunde, dass eine Ersatzoperatorinstanz oder die gleiche Instanz zwar relativ schnell wieder verfügbar ist, dies jedoch so lange dauert, dass es aus Gründen der

Rechtzeitigkeit unmöglich ist, die verlorenen Ereignisse auf dem Splitter aufzubewahren, bis der Ersatz dem System zur Verfügung steht. Die Ereignisse müssen sofort an die anderen Operatorinstanzen verteilt werden, wo sie eventuell vorrangig[1] verarbeitet werden können. Dieses kurzzeitig höhere Nachrichtenaufkommen durch die Neuverteilung muss von den übrigen Operatorinstanzen verarbeitet werden können.

Darüber hinaus wird angenommen, dass der Zeitbedarf für die Bereitstellung von Ersatzinstanzen so hoch ist, dass der Splitter währenddessen weitere, neue Ereignisse aus seinem Eingangsstrom weiter an die $c - 1$ Instanzen verteilt. Dies führt zu einem kurzzeitig noch höheren Netzwerkverkehrsaufkommen an den Operatorinstanzen. Diese Annahme ist sehr realistisch, wenn man die Bereitstellungszeiträume neuer virtueller Knoten in Datenzentren betrachtet. Auch der Neustart einer abgestürzten Instanz ist nicht instantan möglich sondern benötigt in der Praxis etwas Zeit.

6.2 Bestimmung des notwendigen Parallelisierungsgrades c

Wie im vorigen Abschnitt erläutert, kann man die benötigten Ressourcen berechnen. Dabei sollen die Eingangsrate $\lambda = 100$ am Splitter, Verarbeitungsraten $\nu = 15$ in den Operatorinstanzen und die garantierte Wahrscheinlichkeit $p_{soll} = 99$ % konstant sein. Gesucht ist der minimal erforderliche Parallelisierungsgrad c.

Ich berechne zuerst den Parallelisierungsgrad c für den Fall, dass keine Replikation zum Einsatz kommt, bei Ausfällen also immer die vergebenen Selektionen auf die noch vorhandenen Operatorinstanzen zur Berechnung verteilt werden. Tabelle 6.1 auf Seite 53 zeigt die Berechnungen für die einzelnen Konfigurationen im Normalfall ohne Fehler nach den beiden oben genannten Formeln. Für $n \in \{1, 2, 3\}$ sind die Anzahlen der erforderlichen Operatorinstanzen extrem hoch, wobei gleichzeitig die Auslastung ρ sehr niedrig ist. So sind bei der Maximalpufferauslastung von einer Selektion 67 Operatorinstanzen nötig, die aber nur zu 9,99 % ausgelastet sind. Mehr als doppelt so viel Auslastung (21,51 %) erzielt bei etwas weniger als der halben Operatorinstanzanzahl (31) die doppelte Maximalpufferauslastung von $n = 2$. Bei höchstens drei Selektionen in der Warteschlange setzt sich der Trend noch linear fort, es sind 22 Instanzen mit einer Auslastung von 30,30 % notwendig. Mit monoton steigendem n werden sowohl die Auslastung ρ kontinuierlich besser, als auch die Menge der erforderlichen Operatorinstanzen nimmt langsam, aber monoton ab. Bei $n = 4$ sind die 17 benötigten Instanzen zu guten 39 % ausgelastet, bei acht Instanzen für $n = 25$ sogar zu etwa 83 %. Die Auslastung ist bei sehr kleinen n so niedrig und die benötigte Anzahl an Operatorinstanzen gleichzeitig so hoch, dass es sich nicht lohnt, diese Konfiguration in der Praxis zu wählen.

Im Fehlerfall sind jedoch $c - 1$, also im Falle von $c = 4$, z.B. 16 Operatorinstanzen nicht mehr ausreichend, um die Grundlast unter Einhaltung der Garantie abzuarbeiten. Daher

[1] jedenfalls in nach Selektionsnummer aufsteigender Reihenfolge, sodass die abgearbeiteten Selektionen beim Sequenznummerngenerator rechtzeitig ankommen

n	c	ρ	P	OK
1	66	0,1010	0,9898	nein
1	67	0,0995	0,9901	ja
2	30	0,2222	0,9890	nein
2	31	0,2151	0,9901	ja
3	21	0,3175	0,9898	nein
3	22	0,3030	0,9916	ja
4	16	0,4167	0,9874	nein
4	17	0,3922	0,9907	ja
4	18	0,3704	0,9930	ja
4	19	0,3509	0,9947	ja
4	20	0,3333	0,9959	ja
4	21	0,3175	0,9968	ja
4	22	0,3030	0,9974	ja
4	23	0,2899	0,9980	ja
5	14	0,4762	0,9883	nein
5	15	0,4444	0,9923	ja
5	16	0,4167	0,9948	ja
5	17	0,3922	0,9964	ja
5	18	0,3704	0,9974	ja
6	12	0,5556	0,9837	nein
6	13	0,5128	0,9907	ja
6	14	0,4762	0,9944	ja
6	15	0,4444	0,9966	ja
6	16	0,4167	0,9978	ja
7	11	0,6061	0,9818	nein
7	12	0,5556	0,9909	ja
7	13	0,5128	0,9952	ja
7	14	0,4762	0,9974	ja
8	11	0,6061	0,9890	nein
8	12	0,5556	0,9950	ja
8	13	0,5128	0,9975	ja
9	10	0,6667	0,9827	nein
9	11	0,6061	0,9933	ja
9	12	0,5556	0,9972	ja
9	13	0,5128	0,9987	ja
10	10	0,6667	0,9884	nein
10	11	0,6061	0,9959	ja
10	12	0,5556	0,9984	ja
15	8	0,8333	0,9459	nein
15	9	0,7407	0,9918	ja
15	10	0,6667	0,9985	ja
15	11	0,6061	0,9997	ja
20	8	0,8333	0,9783	nein
20	9	0,7407	0,9982	ja
20	10	0,6667	0,9998	ja
25	7	0,9524	0,7188	nein
25	8	0,8333	0,9913	ja
25	9	0,7407	0,9996	ja
25	10	0,6667	1,0000	ja

Tabelle 6.1: Um die 99 %-Garantie im Normalfall unter $\lambda = 100$ und $\nu = 15$ einzuhalten, sind für die einzelnen Warteschlangenfüllstände n teilweise sehr unterschiedliche Parallelisierungsgrade c einzuhalten. ρ und P sind auf vier Nachkommastellen gerundet.

wird als Ausgangsgrundlage der Parallelisierungsgrad c jeweils um eins höher gewählt, für $n = 4$ also $c = 18$ statt $c = 17$. Denn bei insgesamt 18 Operatorinstanzen kann eine davon ausfallen und die restlichen 17 schon einen gleichbleibenden Betrieb gewährleisten. Allerdings ist das gesteigerte Nachrichtenaufkommen durch die Neuverteilung hier noch nicht berücksichtigt. Es muss daher noch überprüft werden, ob auch der sprunghafte Anstieg der Pufferauslastungen durch die Neuverteilung der Selektionen unter Einhaltung der Garantie vollzogen werden kann. Dazu muss die einzuhaltende Pufferauslastung n_{neu} in Wirklichkeit $< n$ sein.

n	c	$\frac{n}{c-1}$	$\lceil \frac{n}{c-1} \rceil$	n_{neu}
1	68	0,0149	1	0
2	32	0,0645	1	1
3	23	0,1364	1	2
4	18	0,2353	1	3
5	16	0,3333	1	4
6	14	0,4615	1	5
7	13	0,5833	1	6
8	13	0,6667	1	7
9	12	0,8182	1	8
10	12	0,9091	1	9
15	10	1,6667	2	13
20	10	2,2222	3	17
25	9	3,1250	4	21

Tabelle 6.2: Die eigentlich einzuhaltenden Pufferauslastungen $n_{neu} = n - \lceil \frac{n}{c-1} \rceil$, um die 99 %-Garantie von $n \in \{1, 2, \ldots, 10, 15, 20, 25\}$ auch im Fehlerfall einzuhalten. Alle Angaben wurden auf vier Nachkommastellen gerundet.

Die ausgefalle Operatorinstanz wird mit 99 %-iger Wahrscheinlichkeit höchstens n Selektionen in ihrem Puffer gehabt haben. Daher müssen n oder weniger Selektionen auf die verbleibenden Instanzen verteilt werden. Um aber auf der sicheren Seite zu bleiben, wird mit genau n Selektionen zur Neuvergabe weitergerechnet. Die rechnerisch entstehenden Neuverteilungen werden ganzzahlig aufgerundet, da nur komplette Selektionen als Ganzes neu vergeben werden, nicht aber nur Teile davon. Dadurch entsteht zwar stellenweise eine kleine Überprovisionierung, jedoch kann nur so die Garantie der Pufferauslastung für alle Instanzen eingehalten werden. Für den Fall $n = 4$ werden dem Ringverteilungsverfahren zufolge vier Operatorinstanzen jeweils eine Selektion zusätzlich erhalten, die anderen 13 gar keine. Tabelle 6.2 zeigt die Bestimmung der eigentlich einzuhaltenden n_{neu}. Wie dort auch sichtlich ist, kann die Formel genau genommen nicht auf $n = 1$ angewendet werden, da die Warteschlangen dann immer leer sein müssten. Da die aktuell verarbeitende Selektion Teil der gefüllten Warteschlange ist, würde das zu einem nicht arbeitenden und damit nicht funktionierendem System führen. Wie oben bereits erwähnt, ist aber für $n \in \{1, 2, 3\}$ kein wirtschaftlich sinnvoller Betrieb möglich, die Werte werden nur der Vollständigkeit halber aufgeführt.

Mit den berechneten n_{neu} kann nun überprüft werden, ob die Garantien eingehalten werden können, vgl. Tabelle 6.3. Wie in der kombinierten Tabelle 6.4 auf Seite 55 ersichtlich ist, ist dies jedoch nur in wenigen Fällen zu leisten. D.h. für die garantierte Maximalpufferauslastung $n = 4$ muss der Parallelisierungsgrad $c = 23$ statt $c = 18$ sein, um auch im Fehlerfall die 99 %-Garantie einzuhalten. Wann immer die Berechnungen zeigen, dass im Fehlerfall die Garantie nicht gewährleistet ist, muss iterativ von einem größeren c aus alles ein weiteres Mal überprüft werden, bis es mathematisch validiert ist.

n	n_{neu}	c_{neu}	ρ	P	OK
1	0	–	–	–	nein
2	1	66	0,1010	0,9898	nein
2	1	67	0,0995	0,9901	ja
3	2	30	0,2222	0,9890	nein
3	2	31	0,2151	0,9901	ja
4	3	18	0,3704	0,9812	nein
4	3	19	0,3509	0,9848	nein
4	3	20	0,3333	0,9877	nein
4	3	21	0,3175	0,9898	nein
4	3	22	0,3030	0,9916	ja
5	4	16	0,4167	0,9874	nein
5	4	17	0,3922	0,9907	ja
6	5	14	0,4762	0,9883	nein
6	5	15	0,4444	0,9923	ja

n	n_{neu}	c_{neu}	ρ	P	OK
7	6	12	0,5556	0,9837	nein
7	6	13	0,5128	0,9907	ja
8	7	11	0,6061	0,9818	nein
8	7	12	0,5556	0,9909	ja
9	8	11	0,6061	0,9890	nein
9	8	12	0,5556	0,9950	ja
10	9	10	0,6667	0,9827	nein
10	9	11	0,6061	0,9933	ja
15	13	9	0,7407	0,9850	nein
15	13	10	0,6667	0,9966	ja
20	17	8	0,8333	0,9624	nein
20	17	9	0,7407	0,9955	ja
25	21	8	0,8333	0,9819	nein
25	21	9	0,7407	0,9986	ja

Tabelle 6.3: Mit den eigentlich einzuhaltenden Pufferauslastungen n_{neu} berechnete Parallelisierungsgrade c_{neu} im Fehlerfall. Auch hier beträgt die Genauigkeit bei allen Angaben vier Nachkommastellen.

			tatsächliches	Unterschied	
n	c	c_{neu}	$c = c_{neu} + 1$	absolut	Faktor
1	68	–	–	–	–
2	32	67	68	36	2,125
3	23	31	32	9	1,391
4	18	22	23	5	1,278
5	16	17	18	2	1,125
6	14	15	16	2	1,143
7	13	13	14	1	1,077
8	13	12	13	0	1,000
9	12	12	13	1	1,083
10	12	11	12	0	1,000
15	10	10	11	1	1,100
20	10	9	10	0	1,000
25	9	9	10	1	1,111

Tabelle 6.4: Tabellen 6.1 und 6.3 kombiniert. Der eigentliche Parallelisierungsgrad c ergibt sich durch die Addition von eins auf c_{neu}. Nur so kann eine Operatorinstanz ausfallen und auch im Fehlerfall die Garantie halten. Die Differenzen zwischen den ursprünglichen c und den tatsächlich zu wählenden c sind teilweise gravierend. Dennoch sind auch einige passgenaue c dabei.

Bis die Ersatzoperatorinstanz zur Verfügung steht, sendet der Splitter aus seinem Gesamteingangsstrom weiter neue Selektionen an die Operatorinstanzen. Jede der verbleibenden Operatorinstanzen erhält daher nun eine neue Eingangsrate $\mu = \frac{\lambda}{c-1}$. Diese ist zwar höher, allerdings muss auch nicht mehr n_{neu} sondern nur noch n eingehalten werden. Anhand von Tabelle 6.1 lässt sich sehen, dass die veränderte Konfiguration problemlos möglich ist. Nicht umsonst wählten wir zu Beginn für jede einzuhaltende Maximalpufferauslastung n den Parallelisierungsgrad c eins größer und mussten dann im weiteren Verlauf mit c_{neu} eventuell noch weiter nach oben gehen. Das Python-Skript, mit dem die Berechnungen für diese hier beschriebenen Fälle durchgeführt wurde, ist auf der CD unter `src/resourceusage.py` zu finden.

Fällt eine Operatorinstanz aus, müssen die anderen mehr Selektionen verarbeiten. Sobald die Ersatzinstanz bereit steht, wird sich die eintreffende Last an den anderen Operatorinstanzen wieder verringern und auf dem Normalwert einpendeln. Die Warteschlangen sind dann mit höchtens n_{neu} Selektionen gefüllt.

Zu beachten ist, dass dieses CEP-System nur einen Operatorinstanzausfall zur selben Zeit toleriert, ohne abgegebene Garantien zu verletzen. Schon die reguläre Last würde ohne Neuverteilung bei $n = 4$ mit 16 Instanzen unter der Pufferauslastungsgarantie nicht mehr zu handhaben sein. Hinzu kommen dann aber auch noch die $2n$ neu zu vergebenden Selektionen der zwei ausgefallenen Instanzen, die an $c - 2$ verbleibende Operatorinstanzen verteilt werden müssen. Meine Aufgabe ist es jedoch, mit lediglich einem Operatorinstanzausfall umgehen zu können, was den Berechnungen zufolge auch gelingen wird.

Die interessanten Fälle, die ich experimentell untersuchen möchte, sind die mit den Maximalpufferauslastungen $n \in \{4, 8\}$. Die meisten Operatorinstanzen werden im Falle von $n = 4$ benötigt. Die zu untersuchenden CEP-Systeme bestehen also aus maximal 23 Operatorinstanzen und den dazugehörigen Laufzeitumgebungen, einem Splitter sowie einem Sequenznummerngenerator, einer Ereignisquelle und einer -senke.

Im Falle von Replikation, bei der alle Selektionen immer von zwei Operatorinstanzen verarbeitet werden, kann – wie in der Aufgabenstellung definiert – höchstens eine der beiden Instanzen ausfallen. Es muss daher bezüglich einer Wiederherstellung nichts weiter veranlasst werden. Insbesondere sind keine Selektionen neu zu vergeben. Aus Tabelle 6.4 ist ersichtlich, dass hier für die Maximalpufferauslastung $n = 4$ der Parallelisierungsgrad $c = 18$ ausreicht. $c = 17$ wäre zwar im Normalfall möglich (vgl. Tabelle 6.1), beim Ausfall einer Operatorinstanz würden die verbleibenden Instanzen jedoch mehr Last abarbeiten müssen, da neue Selektionen trotzdem weiterhin auf die verbleibenden Instanzen verteilt werden. In der hier verwendeten Implementierung gibt es keine festen Replikationspartner, d.h. während eines Ausfalls wechseln diese. Dieser Wert muss allerdings verdoppelt werden, da zwei Operatorinstanzen eine Selektion verarbeiten: $c = 2 \cdot 18 = 36$.

Im Fall der halben Replikation, bei der jede zweite Selektion doppelt vergeben, die übrigen nach dem Wiederherstellungsverfahren gerettet werden, sieht die Berechnung des Parallelisierungsgrads c wie folgt aus: $c = \lceil 1,5 \cdot c_{\text{ohne Replikation}} \rceil$ und somit $c = \lceil 1,5 \cdot 23 \rceil = \lceil 34,5 \rceil = 35$. Hier muss das höhere $c_{\text{ohne Replikation}}$ verwendet werden, da nicht garantiert ist, dass die ausgefallene Operatorinstanz nur replizierte Selektionen in den Warteschlangen hatte. Auch die Annahme, dass nur die Hälfte der Selektionen aus dem Puffer neu vergeben werden müsste,

da die andere Hälfte repliziert worden sei, lässt sich nicht halten. Es kann durchaus vorkommen, dass ausschließlich nicht-replizierte Selektionen in einer Warteschlange enthalten sind. Auch in diesem Verfahren gibt es für die redundante Selektionsvergabe keine festen Paare aus Operatorinstanzen.

6.3 Vorüberlegungen zu möglichen Messabweichungungen beim Durchführen der Experimente

Nachdem die Theorie soweit erläutert ist, stelle ich nun meine Überlegungen zu eventuell abweichenden Messergebnissen vor. Den Gedankenexperimenten zufolge müsste bei einem minimalen, aber ausreichend groß gewählten c – genau den soeben berechneten Werten – die zufällig gemessenen Warteschlangenauslastungen ständig im garantierten Bereich $< n$ liegen. Bei größeren c sollte die Auslastung der Warteschlangen im besten Fall spürbar geringer werden, anderenfalls liegt ein Implementierungsfehler vor oder die anderen Parameter, wie etwa die Eingangsrate λ am Splitter wurden unbemerkt mit verändert.

Der interessantere Fall ist das Verkleinern der ausgerechneten c. Denn dann müsste die Garantie der Einhaltung der maximalen Warteschlangenauslastung über längere Zeit betrachtet verletzt werden. Dies wird vermutlich längere Zeit in Anspruch nehmen, da die Ausfallrate nicht exorbitant hoch ist. Durch Vergrößern der Fehlerrate sollte dies beschleunigt werden können.

Wenn trotz vieler auftretender Fehler die Garantie auch bei verringerten c weiter anhält, könnte das auf die groben und pessimistischen Rundungen bei der mathematischen Berechnung zurückzuführen sein. Wie in Tabelle 6.2 ersichtlich ist, sind die genauen und durchschnittlichen Extraselektionen einer jeden Operatorinstanz beim Wiederherstellen teilweise sehr gering (bei $c = 4$ betragen sie im Durchschnitt 0,2353 Selektionen, bei $c = 5$ sind es immerhin schon 0,3333) und fallen so in der Realität weniger ins Gewicht. Zudem kann sich der Schedulingalgorithmus zeitlich derart geschickt verhalten, sodass zufällig immer die am wenigsten belasteten Operatorinstanzen die meisten – oder überhaupt – erneut abzuarbeitende Selektionen zugewiesen bekommen. In der Implementierung werden die Wiederherstellungsselektionen wie neue Selektionen auf die zur Verfügung stehenden Instanzen der Reihe nach verteilt, natürlich mit Ausnahme der ausgefallen Operatorinstanz. Insofern ist dies recht unwahrscheinlich, aber dennoch nicht ganz unmöglich.

Die zu Beginn sehr groß vorgegebene Wahrscheinlichkeit von 99 % kann dazu führen, dass Auswirkungen eines Operatorinstanzausfalls in den Experimenten nur schwer erkennbar sind. Hier sollten sich jedoch durch die Wahl einer geringeren Wahrscheinlichkeit mehr Schwankungen zeigen lassen.

Sind die auftretenden Ereignisschübe beim Wiederherstellen höher als erwartet, liegt dies höchstwahrscheinlich an der verzögerten Erkennung des Ausfalls. Während dieser Zeit werden der ausgefallenen Instanz weiterhin Selektionen vom Splitter zugeteilt. Diese müssen dann beim Erhalt der Ausfallnachricht ebenso an andere Instanzen außerplanmäßig mitverteilt werden.

6.4 Experimentaufbau und Durchführung

Für die Experimente sind Splitter, Operatorinstanzen und Sequenznummerngenerator sowie Quellen und die Senke als eigene Betriebssystemprozesse realisiert, wobei jedoch eine Operatorinstanz und eine Laufzeitumgebung als ein einzelner Prozess gelten. Die Experimente werden im abteilungseigenen Cluster *NET* (Network Emulation Testbed) ausgeführt. Hierzu stehen mir drei physikalische Knoten zur Verfügung, die jeweils mit acht 1,6 GHz-CPUs sowie 24 GB RAM ausgestatt sind. Laufen also mehr als 24 Prozesse gleichzeitig, müssen sich Prozesse eine CPU teilen. Bei weniger oder genau 24 Prozessen kann jeder davon auf einer eigenen CPU theoretisch unterbrechungsfrei laufen. Somit wird sichergestellt, dass das Betriebssystem mit dem Scheduling möglichst keinen Einfluss auf die Experimentergebnisse nimmt. Allerdings laufen auch andere Systemdienste wie beispielsweise SSH auf den Maschinen, sodass eine komplett unterbrechungsfreie Ausführung nicht ganz gelingen kann. Zumindest ist die Annäherung damit bestmöglich erreichbar.

Um die Experimente sinnvoll analysieren zu können, sollte eine Selektion immer eine feste Größe aufweisen. Hierzu werden von der Selektionsschnittstelle immer drei Ereignisse zu einer Selektion gruppiert. Die Eingangsrate λ am Splitter soll 100 Selektionen pro Zeiteinheit betragen.

Ursprünglich war vorgesehen, fünf Quellen in jedem Experiment insgesamt 8.000 Ereignisse an den Splitter senden zu lassen. Jede Quelle sendet dabei mit einer Exponentialverteilung um den Durchschnittswert von zwei Ereignissen pro Sekunde. Zusammen ergibt das am Splitter eine Eingangsrate von etwa zehn Ereignissen pro Sekunde. Es stellt sich jedoch in einigen Testläufen heraus, dass mehrere Quellen im gleichzeitigen Betrieb teilweise Probleme bereiten. Den Splitter-Protokollen zufolge verklemmt sich eine der Ereignisquellen, was jedoch nicht ganz der Realität zu entsprechen scheint. Wird nach einer Dauer von ca. fünf Minuten das Experiment aufgrund von längerer Inaktivität abgebrochen, zeigt ein eigens für dieses Problem angefertigte Speicherabbild, dass im Splitter von einer der Quellen keine Ereignisse mehr eintreffen. Der dafür vorgesehene Eingangspuffer ist leer, wohingegen die anderen stark gefüllt sind. Die fragliche Quelle hat ihrer erzeugten Protokolldatei zufolge noch sehr viele Ereignisse zu generieren. Jede Quelle selbst besteht im Grunde nur aus den folgenden Zeilen:

```
1 for i in range(1600):
2     sock.send_event()
3     time.sleep(random.expovariate(20) * 10) # Verlangsamung der Experimente um den Faktor 10
```

Insofern kann hier keine Verklemmung auftreten. Dass `random.expovariate(20)` hier einen exorbitant hohen Wert erzeugt, ist unwahrscheinlich im Zusammenhang mit dem vergleichsweise häufigen Auftreten. Das Problem muss also im Splitter vorliegen. Da das Problem nichtdeterministisch auftritt und auf die Schnelle nicht gelöst werden kann, entscheide ich, bei jedem Experiment nur eine einzelne Quelle zu verwenden, die mit fünffacher Rate sendet, also im Mittel 10 Ereignisse die Sekunde erzeugt. So ist für alle Experimente ein gleicher Grundaufbau gewährleistet.

Um Verzögerungen genau messen zu können, leitet jede Operatorinstanz in den Experimenten alle empfangenen Ereignisse einfach an den Splitter weiter. Dabei beträgt die Verarbeitungsrate im Mittel 15 Ereignisse pro zehn Sekunden.

Wie in der Aufgabenstellung beschrieben, sollen ganze Selektionen untersucht werden. Im Prinzip ist es egal, ob Sende- und Empfangsraten in Ereignissen oder Selektionen pro Zeiteinheit angegeben werden, solange das Verhältnis stimmt. Da Selektionen hierbei immer konstant drei Ereignisse umfassen, kann durch die Division der Ereignissenderaten durch drei die tatsächliche Rate in Selektionen ermittelt werden.

Über ein Bash-Skript werden beim Experimentstart die zum Einsatz kommenden Komponenten möglichst gleichmäßig auf den drei physikalischen Knoten verteilt. Nach einer kurzen Wartezeit, um sicherzustellen, dass alle Verbindungen ordnungsgemäß aufgebaut sind, beginnt die Quelle mit der Ereigniserzeugung. Alle Laufzeitumgebungen protokollieren eine Änderung in der Auslastung ihrer Warteschlangen. Sobald eine neue Selektion vom Splitter eintrifft, wird der erhöhte Zähler notiert. Auch wenn das Signal „Korrelation fertig“ der Operatorinstanz eintrifft, wird die nun um eine Selektion verringerte Auslastung festgehalten. Die Selektionsschnittstelle schreibt in die 1.000. Selektion einen Hinweis, der in den Laufzeitumgebungen ausgewertet wird. Erhält eine Laufzeitumgebung diese Notiz, simuliert sie beim Erhalt der „Korrelation beendet“-Nachricht ihrer Operatorinstanz einen einminütigen Ausfall und verwirft alle eintreffenden Ereignisse. Zudem wird die Operatorinstanz gestoppt und der Ausfallzeitpunkt im Protokoll für die Auswertung vermerkt. Nach Ablauf dieser Zeit nimmt sie wieder neue Selektionen vom Splitter an. Sind alle 8.000 Ereignisse versendet und in der Senke angekommen, wird über ein weiteres Bash-Skript das Experiment beendet und die entstandenen Protokolldateien für die Auswertung gesichert. Der Sequenznummerngenerator erkennt Ausfälle, wenn seit dem letzten Empfangszeitpunkt zehn Sekunden vergangen sind. Dieser Wert wurde experimentell als Minimum ermittelt, bei dem in fast allen Experimentläufen keine falsch-positive Erkennung erfolgen. Bei kürzeren Zeiten traten zu viele Falscherkennungen auf, die das Experimentergebnis verfälschten. Ziel der Untersuchung ist jedoch ein einzelner Operatorinstanzausfall. Die Experimentdauer beträgt immer eine knappe Viertelstunde.

7 Lösungsauswertung

Im Folgenden werden die durchgeführten Experimente einzeln ausgewertet. Die Experimente wurden mehrfach im Testbed ausgeführt. Sofern sich keine groben Abweichungen ergeben, wird hier nur ein repräsentativ Ausgewähltes besprochen. Alle Zeitangaben sind relativ zum Experimentbeginn. Sie sind zum besseren Verständnis auf ganze Sekunden gerundet. Operatorinstanzen werden nach ihren TCP-Portnummern benannt, die erste Operatorinstanz ist somit *Instanz 7000*. Rote Markierungen auf der X-Achse (Experimentzeitschiene) machen Instanzausfälle kenntlich. Schwarze Markierungen zeigen die Erkennung des Ausfalls durch den Sequenznummerngenerator, Grüne den Zeitpunkt der Wiederverfügbarkeit aus Sicht des Splitters. Da der Sequenznummerngenerator den Splitter sofort über einen Ausfall benachrichtigt, liegt ihm diese Information nur wenige Millisekunden später ebenfalls vor. Daher kann man sagen, dass beide gleichzeitig Kenntnis über einen Absturz haben.

Es kommen bei den Experimenten drei verschiedene Replikationsstufen r zum Einsatz: Bei $r = 0$ wird eine Selektion nur an eine Operatorinstanz vergeben, beim Ausfall werden die zugeteilten Selektionen durch Neuvergabe wiederhergestellt. Es wird also nicht repliziert. Im Fall von $r = 1$ wird eine Selektion immer an zwei Operatorinstanzen zugeteilt, fällt eine aus, muss keine Wiederherstellung durchgeführt werden. Es kommt also das Replikationsverfahren zum Einsatz. Eine Kombination aus beiden ist $r = 0,5$, also halbe Replikation, bei der jede zweite Selektion redundant an zwei Instanzen überreicht wird. Das jeweilige Zuverlässigkeitsverfahren wird beim Ausfall angewendet.

7.1 Experimentreihe 1

Die nachfolgende Tabelle zeigt die Konfiguration dieses Experiments:

Replikationsstufe r	1
Maximalpufferauslastung n	4
Parallelisierungsgrad c	23
Auslastung ρ	30,30 %
Wahrscheinlichkeit P	99,74 %

In Abbildung 7.1 sind die tatsächlichen Änderungen der Warteschlangenfüllstände dargestellt. Diese Grafik enthält alle 23 Operatorinstanzen und sieht daher auf den ersten Blick etwas unübersichtlich aus. Interessant sind nur die Hochpunkte, nicht die Verläufe der einzelnen Kurven. Zum Zeitpunkt 304 Sekunden nach Experimentbeginn fällt die Operatorinstanz 7011 aus. Sie ist in der Grafik nicht näher zu identifizieren. Zehn Sekunden später detektiert der

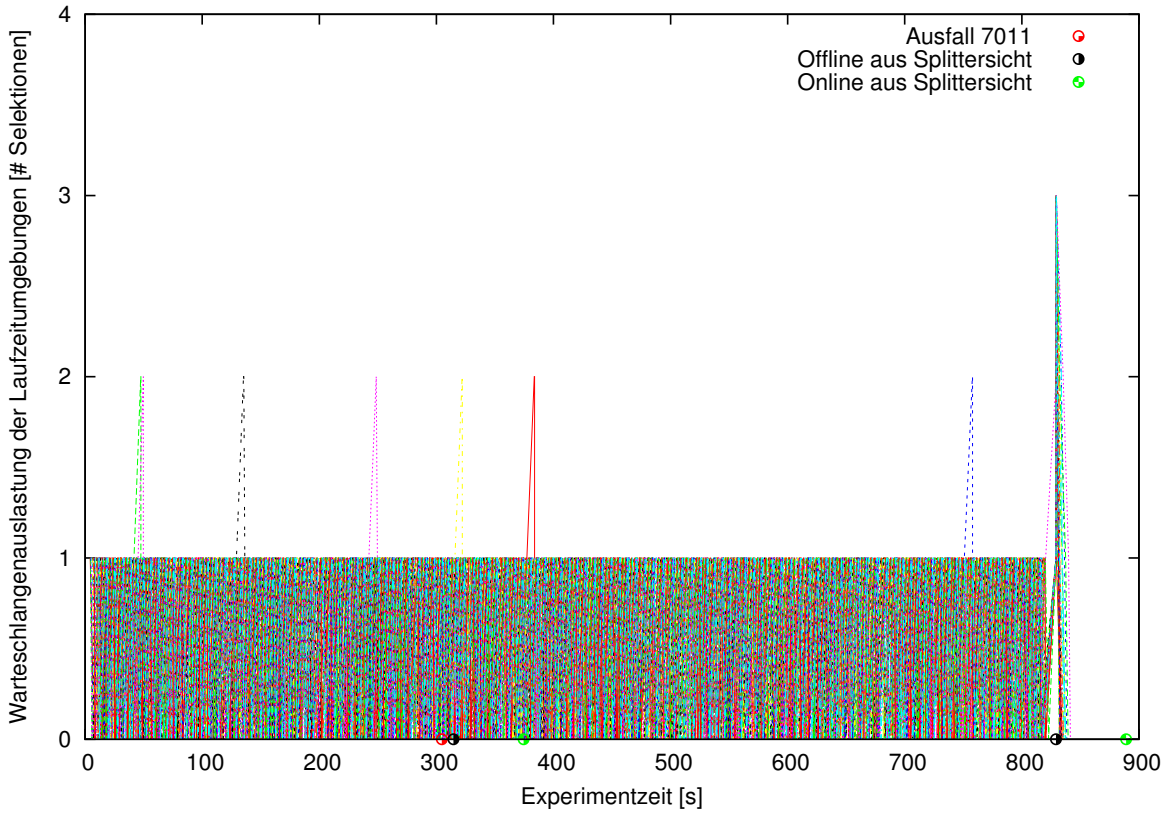


Abbildung 7.1: Warteschlangenauslastungen im Experiment 1 mit $n = 4$, $c = 23$ und $r = 0$.

Sequenznummerngenerator den Ausfall und benachrichtigt den Splitter. Seit dem Zeitpunkt 374 Sekunden ist sie aus Sicht des Splitters wieder online. Kurz vor Experimentende erkennt der Sequenznummerngenerator fälschlicherweise eine Instanz als ausgefallen. Dies zieht größere Auswirkungen nach sich. Der „Ausfall“ wird im Folgenden jedoch ausgeklammert.

Die Pufferauslastung von einer Selektion wird nur äußerst selten überschritten. Es tritt im kompletten Experimentverlauf nur sieben Mal auf, dass die Auslastung auf zwei Selektionen anwächst. Die meisten machen dabei einen periodischen Eindruck, was jedoch höchstwahrscheinlich täuschen mag. Der Tatsache, dass sie fast zyklisch wiederkehren, sollte keine allzugroße Bedeutung beigemessen werden.

Der Ausfall um 304 Sekunden ist überhaupt nicht als solcher in den Pufferfüllständen der anderen Operatorinstanzen zu erkennen. Zwar ist sehr kurz darauf ein Ausschlag auf Selektionen bei einer Instanz zu verzeichnen, dieser kann jedoch genauso gut zufällig entstanden sein, wie dies bei den davor aufgetretenen Spitzen auch der Fall war. Wie durch ein manuelles Nachprüfen in der Protokolldatei des Splitters ersichtlich ist, steht diese Auslastung nicht in direkter Verbindung mit dem Ausfall.

Kurz vor dem Experimentende trat, wie vorhin erwähnt, eine falsche Ausfallerkennung auf. In diesem Fall sind sehr wohl Auswirkungen zu beobachten, die auch deutlich gravierender

ausfallen. Hier trafen wohl zwei Faktoren aufeinander: Zum einen mussten einige Selektionen neu vergeben werden, zum anderen war die simulierte Last auf den anderen Operatorinstanzen vor Erhalt der Neuverteilung recht hoch. Dennoch blieben die Auslastungen aller Warteschlangen unter dem Maximalwert von 4, insbesondere verblieb noch Raum für eine weitere Selektion.

Die Garantie von 99 % in Verbindung mit einem sehr kleinen n hat zur Folge, dass keine aussagekräftigen Beobachtungen in den Experimenten möglich waren. Daher gehe ich direkt auf die nächste Experimentreihe ein.

7.2 Experimentreihe 2

	Experiment 2.1	**Experiment 2.2**	**Experiment 2.3**
Replikationsstufe r	1	2	1,5
Maximalpufferauslastung n	8	8	8
Parallelisierungsgrad c	13	26	20
Auslastung ρ	51,28 %	51,28 %	51,28 %
Wahrscheinlichkeit P	99,75 %	99,75 %	99,75 %

In der Experimentreihe 2 wurden die Maximalpufferauslastungen von 4 auf 8 erhöht. Entsprechend weniger Operatorinstanzen sind dadurch erforderlich. Abbildung 7.2 oben zeigt ein Extrembeispiel der Warteschlangenauslastungen im zweiten Experiment. Auch hier ist wieder ein falsch erkannter Ausfall zu sehen, der gleich zu Beginn des Experiments auftritt. Zu dem Zeitpunkt sind jedoch noch kaum Ereignisse resp. Selektionen[1] im CEP-System, sodass die Wiederherstellung den Experimentverlauf nicht sehr stark negativ beeinflusst. Interessant in diesem Verlauf ist, dass beim Wiederonlinegehen der fälschlicherweise als ausgefallen erkannten Operatorinstanz eine größere und stark verteilte Zunahme der Pufferauslastung aufzutreten scheint. Jedoch stellt sich bei genauerer Betrachtung heraus, dass das Gegenteil der Fall ist. Der einminütige „Ausfall“ führte zu höheren Lasten an den anderen Opertorinstanzen. Als die als ausgefallen deklarierte Instanz in Sekunde 70 wieder im System verfügbar ist, normalisiert sich die Last an den Operatorinstanzen schlagartig.

Nachdem Instanz 7005 nach 319 Sekunden ausfällt, ist im Gegensatz zum ersten Experiment eine deutliche Auswirkung auf die anderen Operatorinstanzen zu erkennen. Während die Auslastung sonst zwischen keiner und einer Selektion hin- und herpendelt, teilweise auch auf zwei, seltener auf drei ansteigt, erreicht der Maximalwert beim Wiederherstellen hier sieben Selektionen im Puffer. Da $n = 8$ ist, ist die Garantie zu jedem Zeitpunkt eingehalten. Zudem beruhigen sich die Pufferstände recht schnell wieder.

Dies zeigt sich auch bei durchschnittlichen Experimentläufen derselben Konfiguration, wie Abbildung 7.2 unten zeigt. Genau genommen ist dort nicht einmal der Ausfall anhand der Pufferfüllstände feststellbar. Hier ist auch eine etwas kürzere Laufzeit ersichtlich. Der

[1] den Protokollen des Splitters zufolge wird Selektion 6 wiederhergestellt

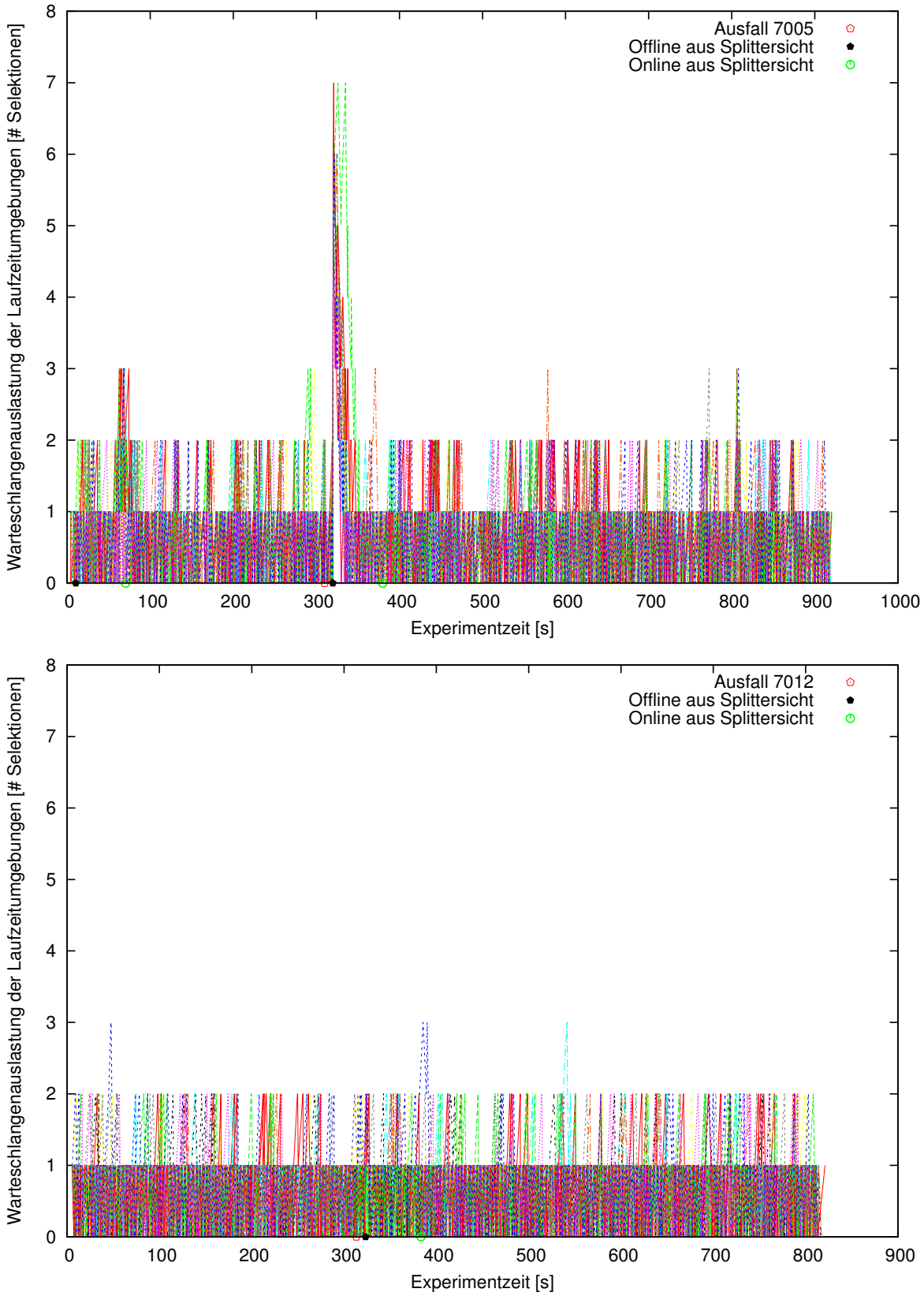


Abbildung 7.2: Warteschlangenauslastungen in den Experimenten 2.1a (oben) und 2.1b (unten) mit $n = 8$, $c = 13$ und $r = 0$.

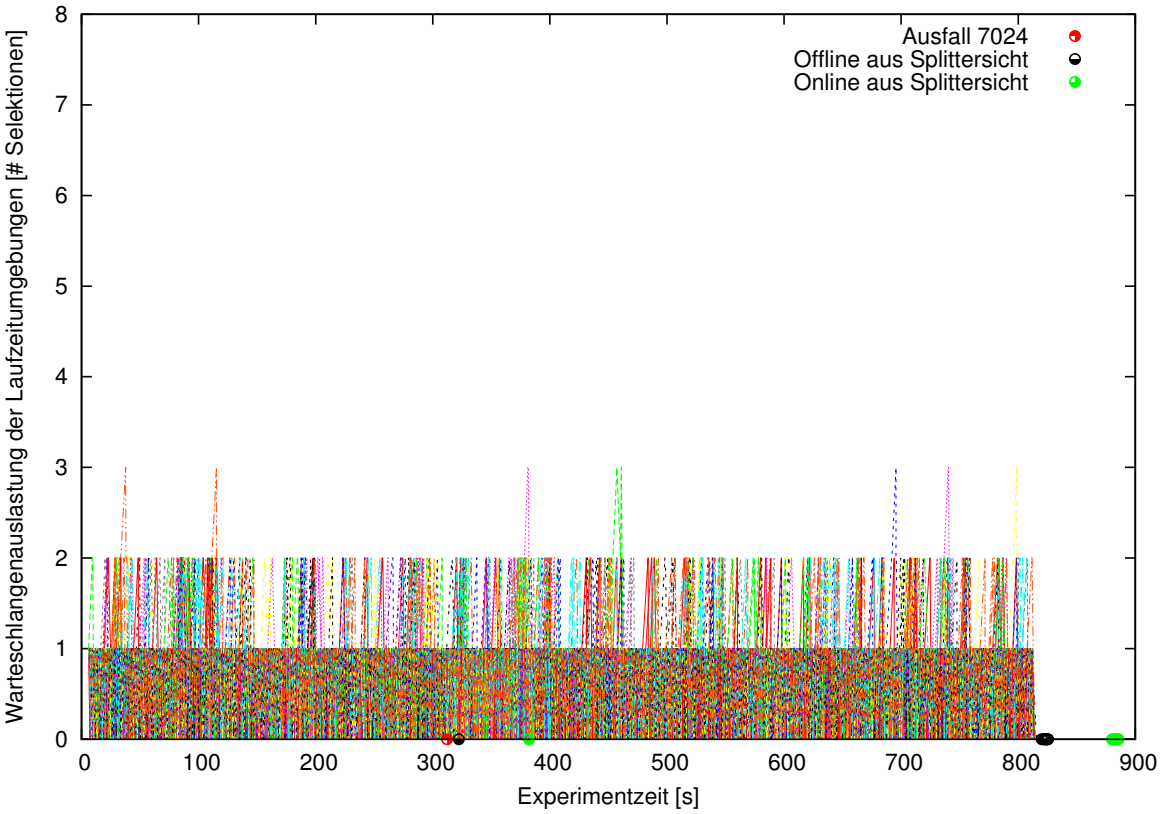


Abbildung 7.3: Warteschlangenauslastungen im Experiment 2.2 mit $n = 8$, $c = 26$ und $r = 1$.

große Ausschlag oben in Experiment 2.1a hat zu Folge, dass sich alles stärker verzögert. Weitere Experimentläufe zur Kontrolle zeigen jedoch immer ein ziemlich ähnliches Bild wie in Experiment 2.1b unten. Die Puffer sind generell mit höchstens zwei Selektionen gefüllt, sehr selten sind drei Selektionen in den Puffern. Höhere Warteschlangenauslastungen konnten nicht mehr erzeugt werden. Es zeigt sich also, dass sich der Pufferfüllgrad in der Regel stark in Grenzen hält und nicht annähernd ausgereizt wird.

Ein fast gleiches Bild der Warteschlangenauslastungen ergibt sich beim selben Experiment mit vollständiger Replikation (vgl. Abbildung 7.3). Dies war auch nicht anders zu erwarten. Dass im Vergleich mit Abbildung 7.2 unten öfters Pufferstände der Größe 3 erreicht werden, ist Zufall, was Kontrollläufe auch bestätigen. Auch schwankt die Dauer des Experimentlaufs immer um den Wert 800 Sekunden, mal sind die Schwankungen größer, mal kleiner. Ergebnis ist hier, dass die 99 %-Garantie problemlos eingehalten wird.

Auch halbe Redundanz (s. Abbildung 7.4) ist nicht von der Verteilung an immer nur eine (Experiment 2.1) oder immer an zwei Operatorinstanzen (Experiment 2.2) zu unterscheiden. Dies gilt für alle untersuchten Konfigurationen. Im Zusammenhang mit Pufferauslastungen macht es keinen großen Unterschied, welches Zuverlässigkeitsverfahren verwendet wird. Ein näheres Betrachten der Protokolldateien gibt allerdings dennoch Aufschluss über die

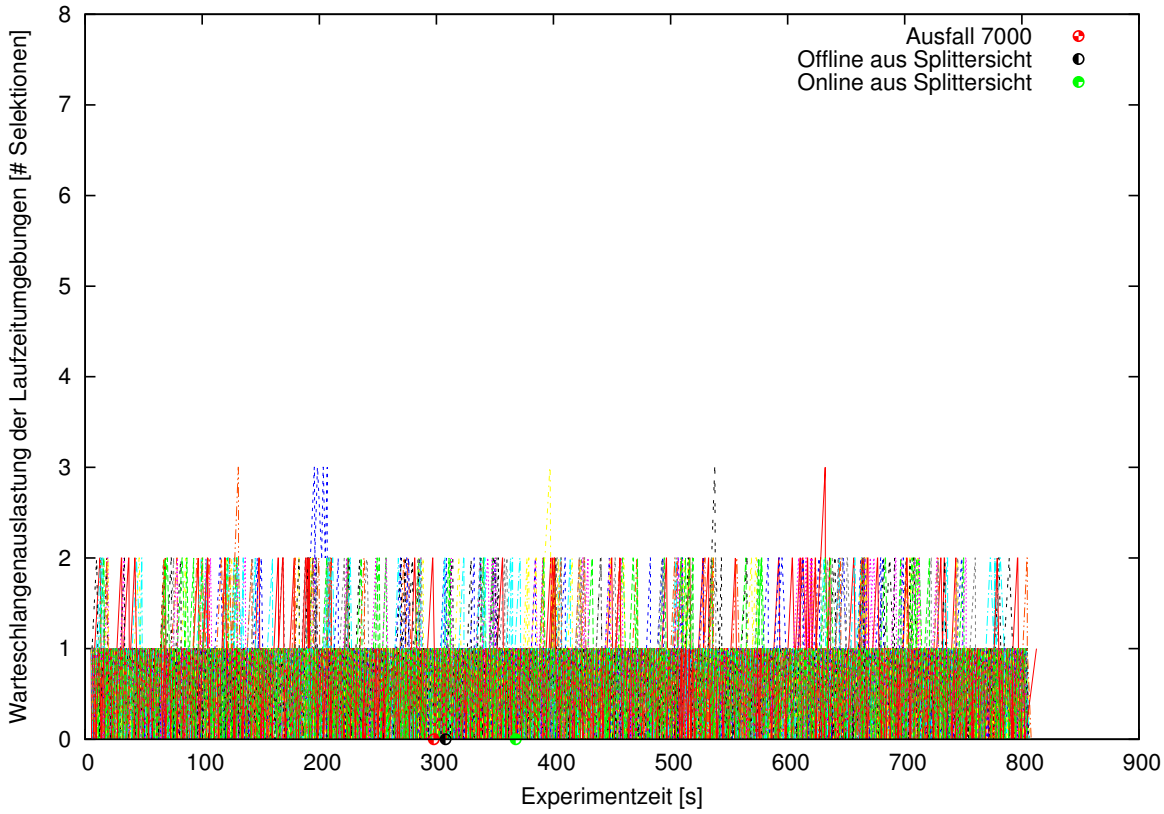
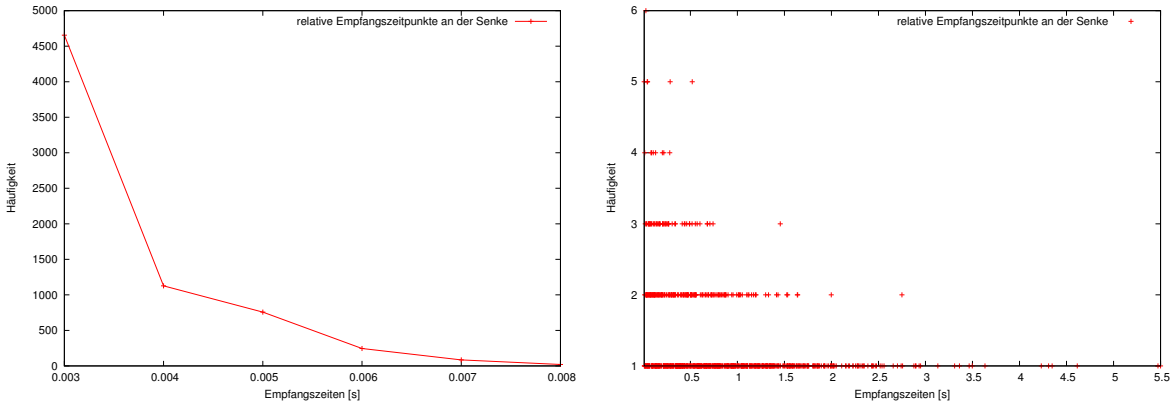


Abbildung 7.4: Warteschlangenauslastungen im Experiment 2.3 mit $n = 8$, $c = 20$ und $r = 0,5$.

Verzögerungszeit der Selektionen in einem Operator. Wenn eine Selektion repliziert wurde, entsteht keinerlei Verzug. Muss sie jedoch wiederhergestellt werden, verschiebt sich der Zeitpunkt des Verlassens des Operators um zehn Sekunden. Dies ist genau die Zeit, die der Sequenznummerngenerator benötigt, um einen Ausfall zu erkennen. Die Wiederherstellung der fehlenden Selektionen geschieht somit in nahezu Nullzeit. Also liegt dieser beobachtete Effekt an der konkreten Implementierung der Experimente, nicht am Verfahren selbst. Es kann daher also *nicht* daraus geschlossen werden, dass Replikation *bedeutend* schneller ist.

Die Verzögerungszeiten sind hierbei sehr kurz (vgl. Abbildung 7.5a). Mit relativen Empfangszeitpunkten ist im Folgenden die Dauer gemeint, die zwischen zwei Empfangszeitpunkten liegt. Nur 13,8 % der relativen Empfangszeitpunkte an der Senke liegen dabei über 8 Millisekunden, 82,2 % der Ereignisse werden zwischen 3 und 8 Millisekunden nach den zuletzt erhaltenen Ereignissen empfangen (s. Abbildung 7.5b). Gerade einmal 3,5 % aller relativen Empfangszeitpunkte liegt über einer Sekunde. Mehr als 5,5 Sekunden wird nur beim Wiederherstellen der fehlenden Selektion einmal erreicht. Die Verzögerung liegt hierbei bei 10,5 Sekunden (in Abbildung 7.5b nicht dargestellt). Dies liegt daran, dass die Ereignisse vom Sequenznummerngenerator in Schüben ausgeliefert werden. Eine zeitliche Betrachtung der relativen Empfangszeitpunkte zeigt in Abbildung 7.6, dass diese über die ganze



(a) Relative Empfangszeitpunkte zwischen 3 und 8 Millisekunden.

(b) Relative Empfangszeitpunkte zwischen 9 Millisekunden und 5,5 Sekunden.

Abbildung 7.5: Relative Empfangszeitpunkte an der Senke während des Experiments 2.1b.

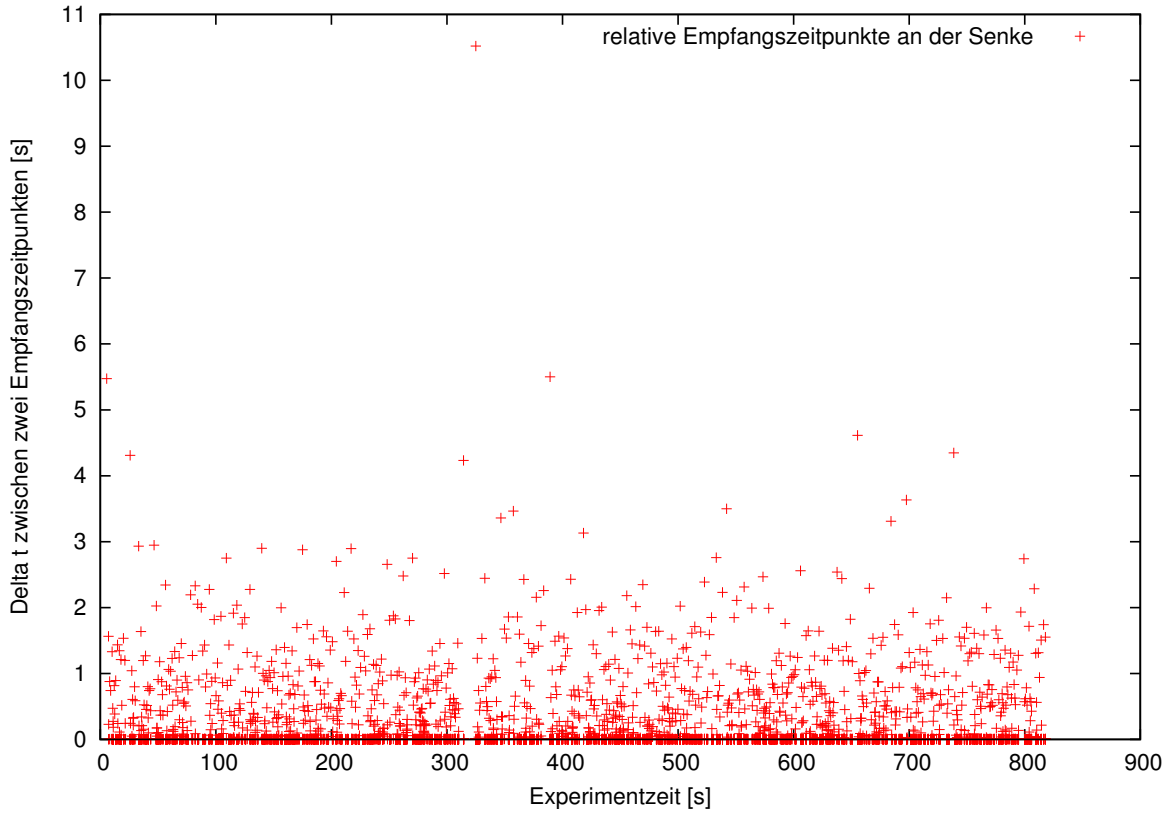


Abbildung 7.6: Relative Empfangszeitpunkte an der Senke über die gesamte Dauer des Experiments 2.1b.

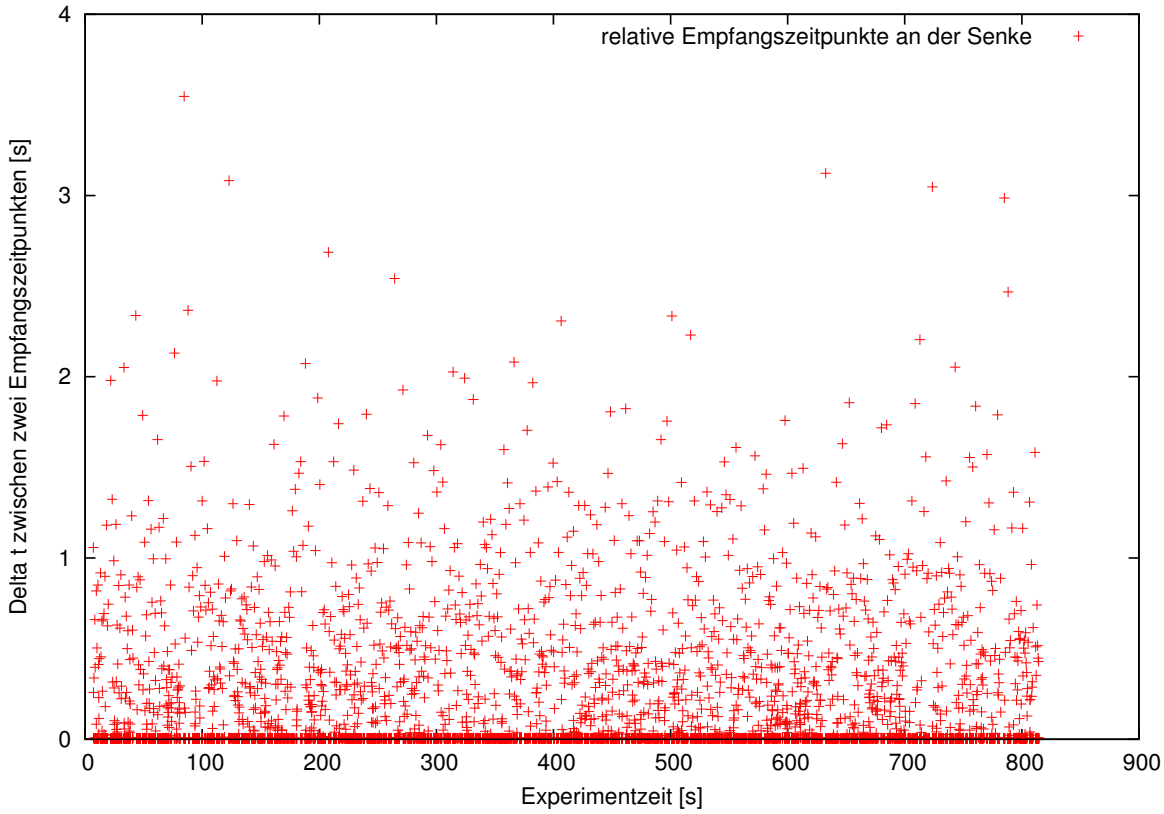


Abbildung 7.7: Relative Empfangszeitpunkte an der Senke über die gesamte Dauer des Experiments 2.2.

Experimentdauer hinweg gleichverteilt sind. Die in diesem Graphen erste zu sehende 5,5-Sekundenverzögerung kommt durch die kurze Wartezeit der Quelle am Experimentbeginn zustande. Abbildung 7.7 zeigt die gleiche Verteilung im Experiment 2.2, als Selektionen redundant vergeben wurden. Hier liegen die größten Differenzen unter 4 Sekunden. In allen weiteren Experimentreihen lassen sich fast identische Werte ermitteln. Sie schwanken dabei höchstens im Halbsekundenbereich, häufig sind die Maximalwerte – mit Ausnahme der betroffenen Selektion – sogar leicht unterhalb diesen hier vorgestellten Messergebnissen. Da der Ausfall in manchen Fällen schon während dem Serialisieren von vorigen Selektionen erkannt wird, liegt die Verzögerung der fehlenden Selektion teilweise unter neun Sekunden. In den folgenden Experimentreihen wird daher auf die relativen Empfangszeitpunkte nur in Ausnahmefällen näher eingegangen.

Nach einer Analyse der Protokolldateien von Quelle und Senke lässt sich eine mit der Zeit größer werdende Differenz von Absendezeitpunkten an der Quelle und Empfangszeitpunkten an der Senke feststellen. Beim Wiederherstellungsverfahren beträgt die Dauer zwischen Versand und Empfang beim ersten Ereignis noch 200 Millisekunden, so sind es beim 8.000. Ereignis am Ende 31 Sekunden. Die Fehlererkenndauer von zehn Sekunden spiegelt sich auch in dieser Analyse wider. So sind zwischen dem letzten empfangenen Ereignis vor dem Fehler und dem

ersten nach der wiederhergestellten Selektion etwa zehn Sekunden vergangen. Ansonsten ist der Anstieg der festgestellten Verzögerung linear. Auch bei der Replikation in Experiment 2.2 ist ein ähnlicher Effekt zu beobachten, der allerdings nicht ganz so stark ausfällt, hier beträgt die Differenz zum Schluss nur 6 Sekunden. Über mehrere Experimentläufe verschiedener Konfigurationen hinweg treten stets Verzögerungen gleicher Größenordnungen auf. Dieses ansteigende Verhalten liegt mit Sicherheit darin begründet, dass die Operatorinstanzen die empfangenen Ereignisse einfach alle weiterleiten und nicht aggregiert weitersenden. Zudem schreiben sie einige Metainformationen für die Fehlersuche mit hinein. Somit bleibt nicht nur die Anzahl der Ereignisse gleich, auch sind die Gesendeten etwa doppelt so groß wie die empfangenen Ereignisse. Dies muss sich zwangsläufig über die Zeit bemerkbar machen. Nun könnte man annehmen, dass die Replikation für deutlich weniger Verzögerungszeiten als die Wiederherstellung sorgt, dem ist aber nicht so. Dass Replikation und Wiederherstellung so unterschiedliche Differenzen aufweisen, liegt einerseits an der 10 Sekunden dauernden Fehlererkennung, andererseits an dem etwas ungeschickten Versuchsaufbau, der die Replikation begünstigt. Die Operatorinstanzen arbeiten autonom und bestimmen für jedes Ereignis einer Selektion ihre wahrscheinlichkeitsverteilten Verarbeitungszeiten unabhängig voneinander. Im Falle ohne replizierte Informationen entsteht nur eine Verarbeitungszeit pro Ereignis. Bei der Replikation erzeugen aber zwei verschiedene Operatorinstanzen für dasselbe Ergebnis zwei unterschiedliche Verzögerungen. Der Sequenznummerngenerator muss aber nur auf die kürzere der beiden warten, wohingegen im zuerst genannten Fall immer auf die potentiell größere Verarbeitungszeit zu warten ist. Anhand der Protokolldatei des Sequenznummerngenerators lässt sich diese Erklärung gut nachvollziehen und bestätigen.

7.3 Experimentreihe 3

	Experiment 3.1	**Experiment 3.2**	**Experiment 3.3**
Replikationsstufe r	1	2	1,5
Maximalpufferauslastung n	8	8	8
Parallelisierungsgrad c	8	16	12
Auslastung ρ	83,33 %	83,33 %	83,33 %
Wahrscheinlichkeit P	80,62 %	80,62 %	80,62 %

In den vorigen Experimenten war die Wahrscheinlichkeit zum Einhalten der Pufferauslastungen mit über 99 % zu hoch, als dass jemals Überschreitungen der Pufferauslastungen in weniger als 1 % der Fälle sichtbar gewesen wären. Daher wird in der dritten Experimentreihe durch Herabsetzen der Parallelisierungsgrade auch die Wahrscheinlichkeit reduziert, dass Pufferauslastungen ≤ 8 bleiben. Die rechnerische Auslastung der Operatorinstanzen steigt dabei von 51 % in der letzten Experimentreihe auf nun 83 %. Mit mehr als 80 % ist die damit erzielte Wahrscheinlichkeit vergleichsweise gering und reicht aus, ein Überschreiten der Maximalauslastungen auszulösen (s. Abbildung 7.8). Drei Selektionen in den Warteschlangen sind hierbei schon ziemlich normal (in vorigen Experimenten waren es lediglich zwei), teilweise steigt die Pufferauslastung auch auf bis zu vier. Im Fehlerfall wird die Grenze von $n = 8$ von einer Operatorinstanz um höchstens zwei Selektionen für eine Dauer von 71 Sekunden

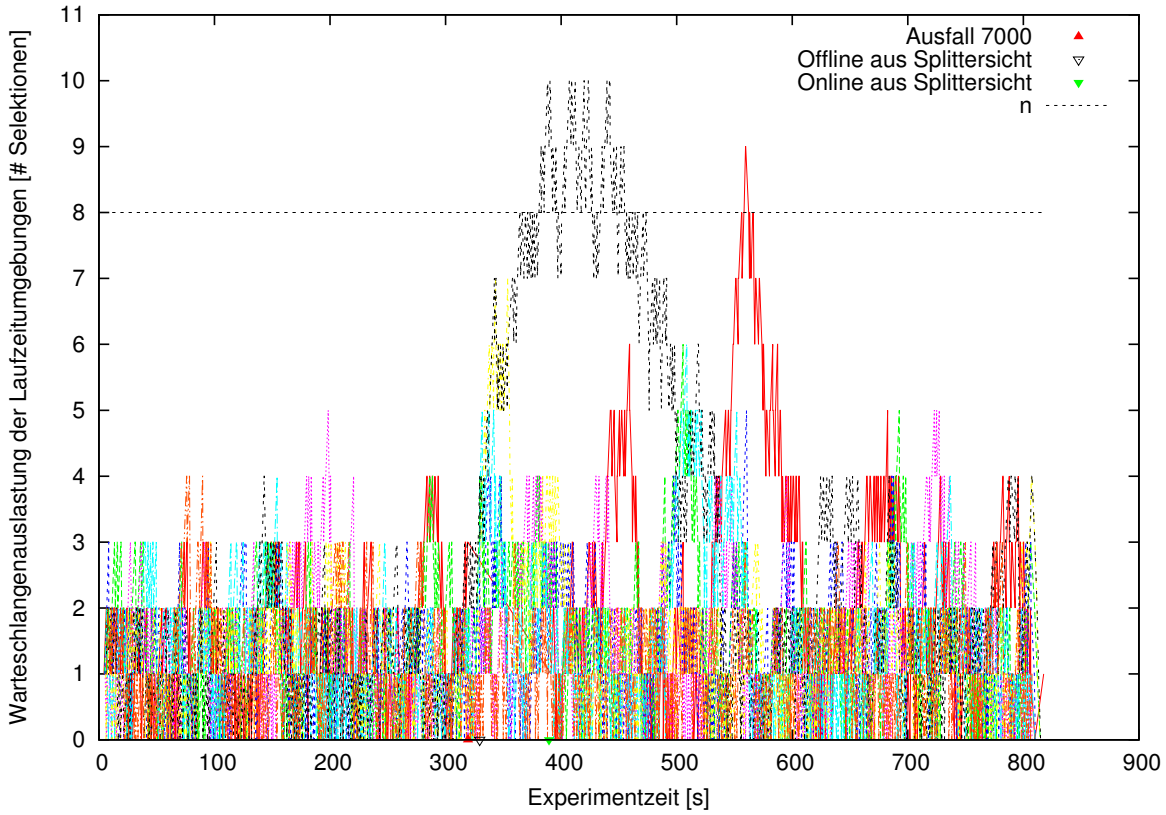
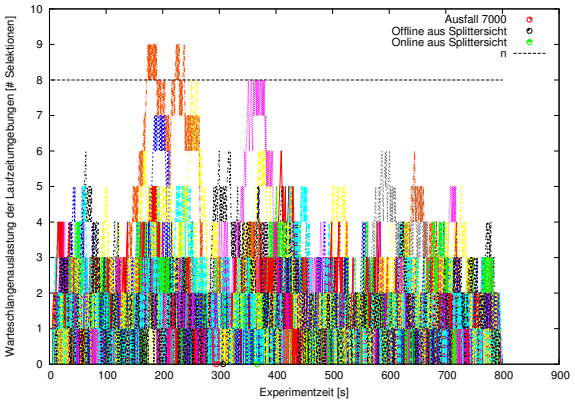
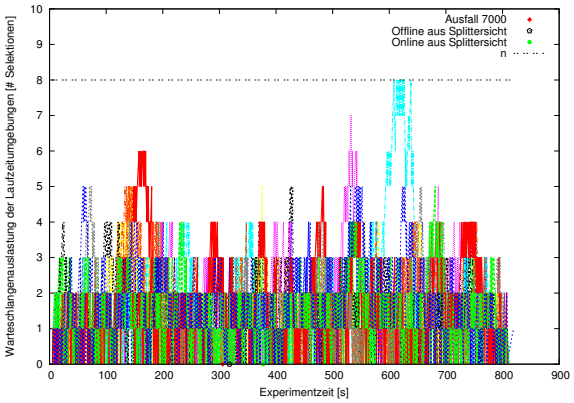


Abbildung 7.8: Warteschlangenauslastungen im Experiment 3.1 mit $n = 8$, $c = 8$ und $r = 0$.



(a) Experiment 3.2: $n = 8$, $c = 16$ und $r = 1$



(b) Experiment 3.3: $n = 8$, $c = 12$ und $r = 0,5$

Abbildung 7.9: Warteschlangenauslastungen in den Experimenten 3.2 und 3.3.

überschritten. Ebenso ist im wieder eingekehrten Normalbetrieb die Überschreitung um eine Selektion einmal kurzzeitig der Fall.

Dass es tatsächlich nicht nur im oder nach einem Fehlerfall zur Überschreitung des Grenzwerts kommt, zeigt auch das Experiment 3.2 mit kompletter Replikation. In Abbildung 7.9a ist gut zu erkennen, dass schon vor dem Ausfall der Operatorinstanz die Warteschlangen einer Operatorinstanz auf neun Selektionen angewachsen sind. Damit ist der erlaubte Maximalwert überschritten, was jedoch wegen der verringerten Garantie in Ordnung ist. Der Ausfall kann in diesem Experimentlauf sogar innerhalb der Schranke $n = 8$ abgefangen werden.

Wie Experiment 3.3 in Abbildung 7.9b gut zeigt, muss bei einer niedrigeren Garantie die Auslastung der Puffer nicht zwangsläufig verletzt werden. Im hier abgebildeten Experimentlauf mit einer Mischung der beiden Zuverlässigkeitsverfahren ist sogar der Ausfall im Graphen nicht erkennbar. Hier würde man darauf tippen, dass die fragliche Selektion repliziert wurde. Tatsächlich ist anhand der Protokolle des Splitters jedoch ablesbar, dass genau dies nicht der Fall war, sondern sogar sieben Selektionen wiederhergestellt werden mussten. Das bedeutet, dass alle verbleibenden Operatorinstanzen eine zusätzliche Selektion überreicht bekamen. Die geforderte, maximale Puffergrenze wird zwar exakt erreicht, jedoch nicht überschritten.

Interessant an dieser Experimentreihe ist, dass trotz einer vergleichsweise recht geringen Garantie von knapp über 80 % die Maximalpufferauslastung sehr gut eingehalten wird. In allen durchgeführten Experimentläufen waren nie mehr als zwei von acht Operatorinstanzen gleichzeitig über dem Grenzwert (in den Abbildungen 7.8 und 7.9a ist nur eine Instanz zur selben Zeit darüber). Auch ist die Dauer der Überschreitung meist überschaubar.

7.4 Experimentreihe 4

Replikationsstufe r	1
Maximalpufferauslastung n	8
Parallelisierungsgrad c	7
Auslastung ρ	95,24 %
Wahrscheinlichkeit P	35,54 %

Möchte man als Betreiber seine Knoten möglichst effizient auslasten, zeigt Experiment 4a in Abbildung 7.10 sehr gut, was bei einer 95-prozentigen Auslastung der Knoten geschieht. Im Vergleich zur vorigen Experimentreihe wurde für den Fall ohne Replikation lediglich eine Operatorinstanz weggenommen. Die Wahrscheinlichkeit beträgt im Rechenmodell gerade noch 35 %, dass der Warteschlangenfüllgrad von $n = 8$ Selektionen eingehalten werden kann. Im Normalbetrieb ist dies in diesem Experimentlauf anfangs exakt einzuhalten, doch im Fehlerfall schießen die Auslastungen teils drastisch in die Höhe. Der Maximalwert beträgt hier 20 Selektionen. Dies ist mehr als das doppelte der einzuhaltenden Maximalpufferauslastung. Gut zu erkennen ist hierbei auch, dass während des Ausfalls (Sekunden 316 – 376) keine der Operatorinstanzen einen leeren Puffer hat. Die Auslastung beruhigt sich erst wieder, nachdem die ausgefallene Operatorinstanz dem CEP-System wieder zur Verfügung steht. An diesem Graphen ist zudem ersichtlich, dass bei weniger Instanzen eine normaldurchschnittliche

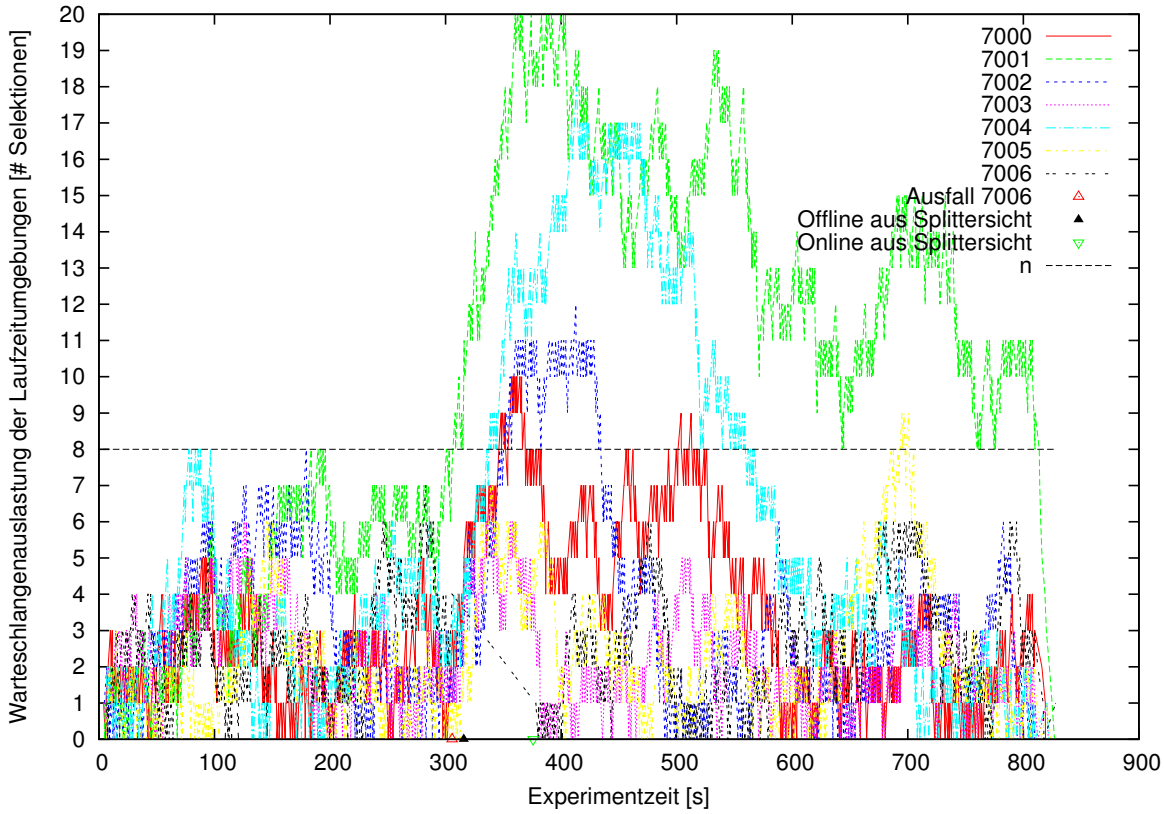


Abbildung 7.10: Warteschlangenauslastungen im Experiment 4.1a mit $n = 8$, $c = 7$ und $r = 0$.

Verteilung erst später wiederhergestellt werden kann. Instanz 7001 (grün dargestellt) erholt sich sogar erst am Ende des Experiments, wenn die Quelle keine Ereignisse mehr erzeugt und daher vom Splitter auch keine weiteren Selektionen mehr vergeben werden. In anderen Testläufen werden teilweise nur Warteschlangenauslastungen von 13 oder auch weniger Selektionen gemessen.

In Abbildung 7.11 ist der Lauf des Experiments 4.1b dargestellt. Er zeigt die Erkennung von zwei Ausfällen, wobei nur der erste zum Zeitpunkt 315 Sekunden nach Experimentbeginn der echte Ausfall ist. Nach einer knappen halben Minute tritt zum Zeitpunkt 343 eine falsch-positive Ausfallerkennung auf. Somit stehen zu diesem Zeitpunkt zwei von sieben Operatorinstanzen dem CEP-System nicht mehr zur Verfügung. Wie im Lösungsansatz erwähnt, kann das System immer nur mit einem Ausfall zur selben Zeit umgehen. Da hier jedoch auch die Garantie mit nur 35 % sehr gering ist, werden die Puffer der verbleibenden fünf Instanzen rasant mit neuen Selektionen aufgefüllt. Der größte erreichte Füllstand beträgt 28 Selektionen und damit das 3,5-fache des Einzuhaltenden. Zu erkennen ist auch, dass sich das CEP-System hinsichtlich Pufferfüllständen nicht mehr regenerieren kann. Erst als die Versorgung mit neuen Selektionen aufhört, können die Instanzen ihre Puffer nach und nach leeren. Die Verzögerung im Fehlerfall beträgt in diesem Experiment bei hoher Last

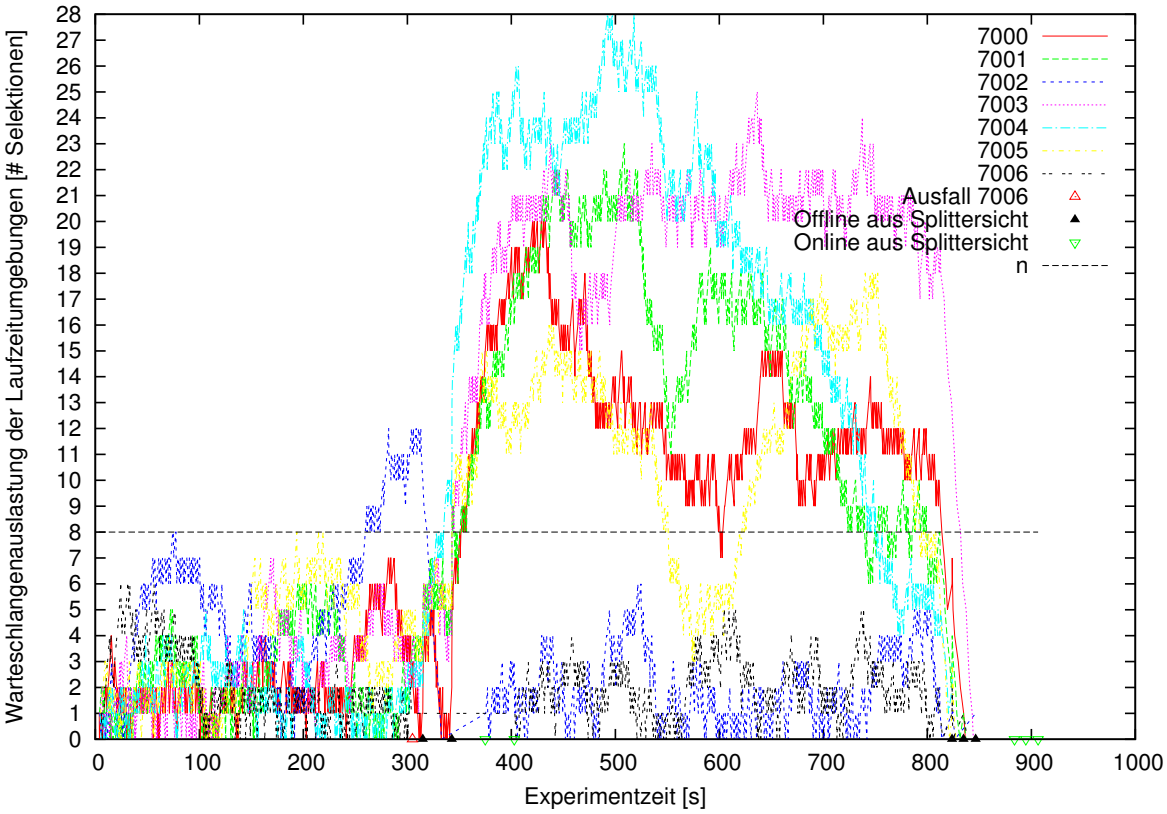


Abbildung 7.11: Warteschlangenauslastungen im Experiment 4b mit $n = 8$, $c = 7$ und $r = 0$.

sogar einmal bis zu 24 Sekunden. Kurz vor Ende verzögern sich drei Selektionen, sodass der Sequenznummerngenerator die entsprechenden Operatorinstanzen als ausgefallen erkennt. Sehr interessant ist jedoch, dass die Laufzeit dieses Experiments nicht deutlich länger ist. Mit 843 Sekunden ist die Experimentdauer immer noch im üblichen Bereich der Schwankungen.

7.5 Experimentreihe 5

Um zu zeigen, dass eine 100-prozentige Auslastung der Operatorinstanzen im Normalbetrieb tatsächlich unter keinen Umständen verwendet werden sollte, ist die Konfiguration für die fünfte Experimentreihe angepasst worden. Dazu wurden die Verarbeitungsraten ν der Opatorinstanzen von 15 auf 20 erhöht. So können beim Verfahren ohne Replikation fünf Instanzen im Normalbetrieb komplett zu 100 % ausgelastet werden. Bei $\nu = 15$ und $c = 6$ würden die Instanzen mit 111 % sowieso überlastet sein. Das Ergebnis ist in Abbildung 7.12 zu sehen. Diese Grafik zeigt den Betrieb ohne Replikation. Unschwer zu erkennen sind die ständig ansteigenden Pufferauslastungen, nur selten ist ein Puffer leer. Mit dem Ausfall der stark ausgelasteten Instanz 7000 (rot) müssen sehr viele Selektionen neu vergeben werden. Diese elf müssen auf vier verbleibende Operatorinstanzen verteilt werden, was zu einem sehr hohen

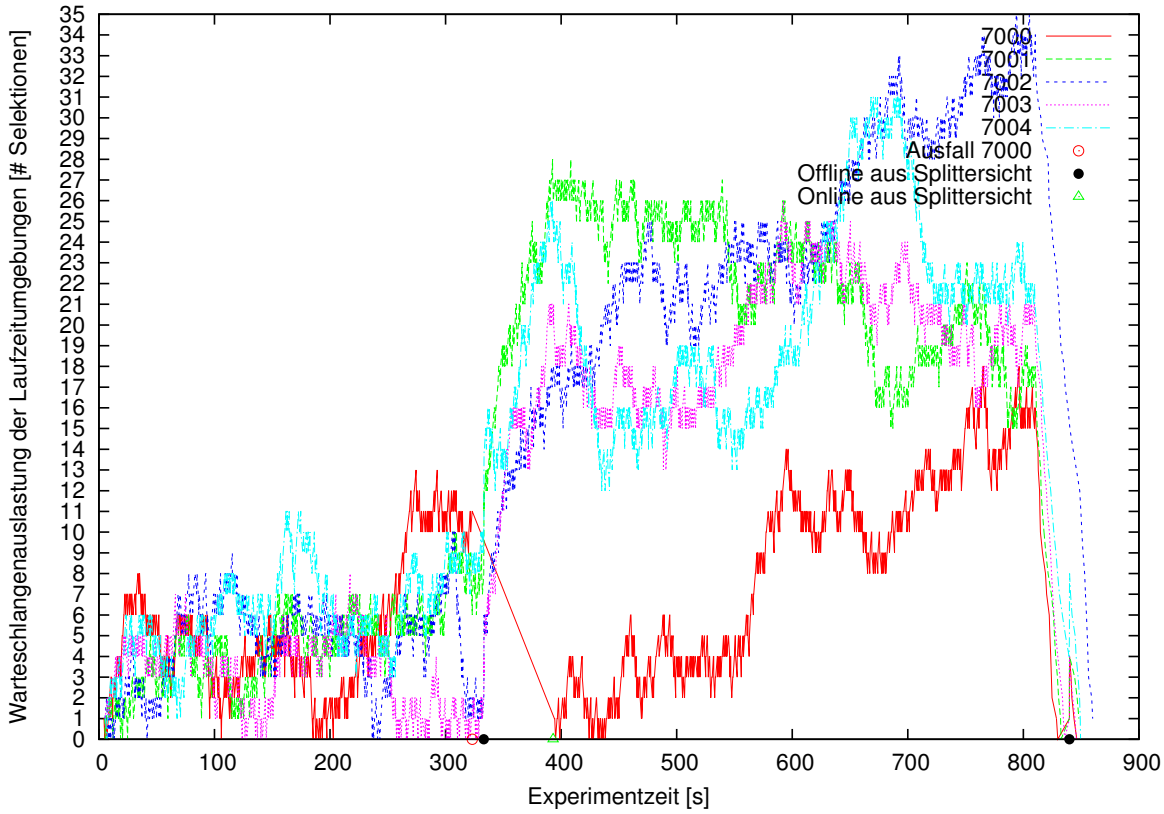


Abbildung 7.12: Warteschlangenauslastungen im Experiment 5.1 mit $\nu = 20$, $c = 5$ und $r = 0$.

Anstieg der Warteschlangen führt. Der Operator hat in diesem Fall keine Möglichkeit sich zu regenerieren. Unabhängig von einer konkreten Maximalpufferauslastung n ist zu sagen, dass es nur eine Frage der Zeit ist, bis diese Schranke überschritten wird. Die Pufferauslastungen erholen sich auch hier erst wieder, sobald der Nachschub an neuen Selektionen versiegt.

Die mit vollständiger ($r = 1$) und halber Replikation ($r = 0,5$) durchgeführten Messreihen zeigen das gleiche Verhalten. Abbildung 7.13 stellt exemplarisch den Ablauf mit halber Replikation dar. Da 5 eine ungerade Zahl ist, musste die anderthalbfache Anzahl der Operatorinstanzen bei diesem Verfahren von 7,5 auf 8 erhöht werden, was die Steigung der einzelnen Kurven im Vergleich zu Abbildung 7.12 etwas abflachen lässt.

7.6 Schlussfolgerungen und Ausblick auf Erweiterungen

Die Auswertung der durchgeführten Experimente zeigt, dass der Lösungsansatz korrekt war. Der mithilfe der Warteschlangentheorie berechnete Parallelisierungsgrad führt tatsächlich dazu, dass die Auslastungen der Warteschlangen entsprechend eingehalten werden. Dies ist auch der Fall, wenn eine Operatorinstanz ausfällt. Bei geringeren Parallelisierungsgraden

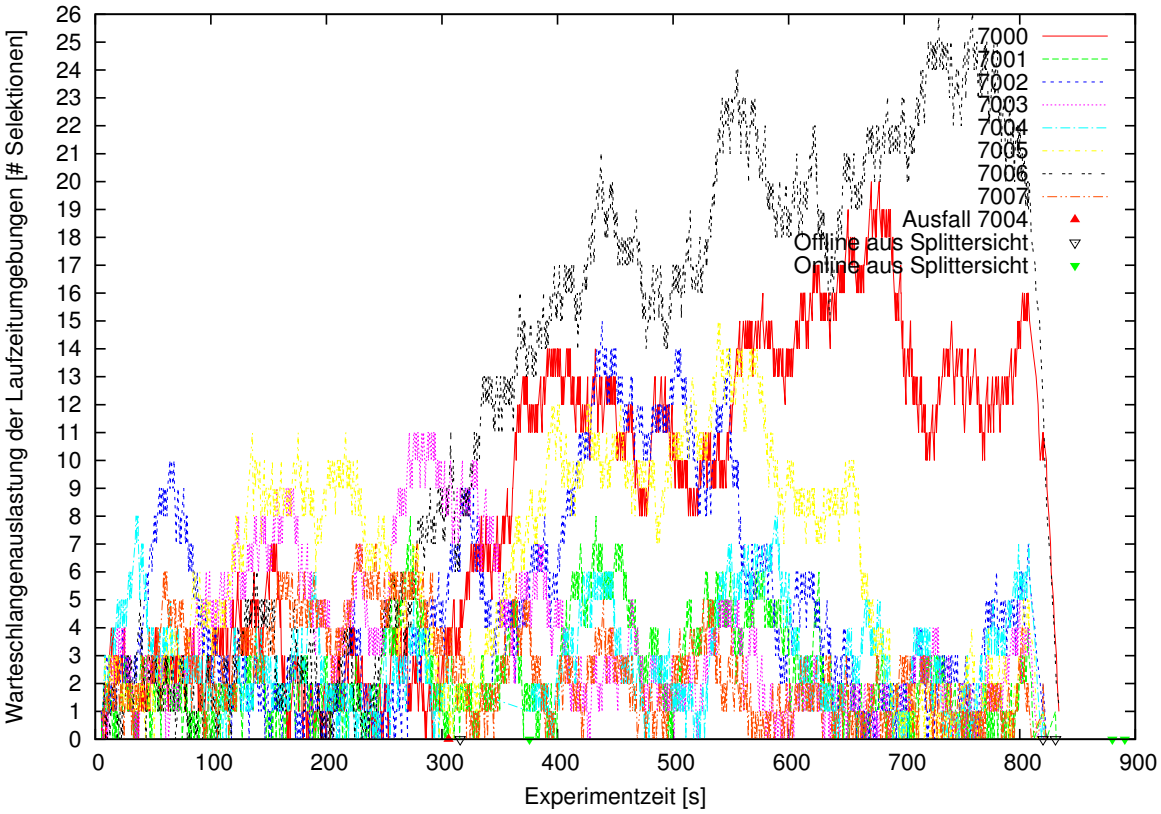


Abbildung 7.13: Warteschlangenauslastungen im Experiment 5.2 mit $\nu = 20$, $c = 8$ und $r = 0,5$.

werden diese Garantien jedoch nicht eingehalten, obwohl die Auslastungen der Operatorinstanzen unter 100 % liegen.

Damit verbunden ist auch die Erkenntnis, dass die Operatorinstanzen nur sehr gering ausgelastet werden dürfen, um auch im Fehlerfall die Maximalauslastung der Warteschlangen mit Sicherheit einzuhalten.

Weiterhin zeigen die Untersuchungen, dass die Verwendung von Replikation im Gegensatz zur Wiederherstellung von Selektionen keine nennenswerte Vorteile bietet. Zwar bringt sie in den hier konkret untersuchten Fällen immer einen Vorsprung in Höhe der Erkennungszeit von Ausfällen, jedoch kann durch den Einsatz einer verzögerungsfreien Ausfallerkennung mit dem Wiederherstellungsverfahren das gleiche Ergebnis erzielt werden. Die eigentliche Wiederherstellung der Selektionen, also die Neuvergabe, ist mit kaum einer Verzögerung verbunden. Mit der Replikation sind daher sogar Nachteile verbunden. So müssen deutlich mehr Ressourcen – je nach Verfahren anderthalb oder doppelt so viele – bereitgestellt werden. Auch fallen die immer doppelt vergebenen Selektionen mehr ins Gewicht als die nur bei der Wiederherstellung neuzuverteilenden Selektionen. Es ist also wirtschaftlich besser, das

Wiederherstellungsverfahren zu verwenden ohne zeitliche Einbußen in Kauf nehmen zu müssen – vorausgesetzt, die Ausfallerkennung ist schnell genug.

8 Related Work

Nicht alle Verfahren bieten eine vollständige Wiederherstellung und damit absolut korrekte Auslieferung der erzeugten komplexen Ereignisse. Im Fehlerfall versuchen diese Systeme etwa weiterhin Ergebnisse zu erzeugen, die zwar nicht ganz korrekt sind, aber für den Konsumenten dennoch hilfreich sein können [BBJ+08, JSGAW09]. Zudem gibt es den Ansatz, bereits ausgelieferte und als vorläufig markierte Ergebnisse mithilfe von Korrekturereignissen zu revidieren [BFSF08]. In dieser Arbeit werden jedoch immer vollständig korrekte Ergebnisse an den Konsumenten ausgeliefert.

Viele Parallelisierungskonzepte betrachten die Zuverlässigkeit nicht im Sinne von Rechtzeitigkeit. Der hier verwendete Ansatz integriert jedoch beide Eigenschaften, er stellt verlorene Ströme rechtzeitig wieder her. Dies ist in anderen Ansätzen zur Parallelisierung [Hir12, BDWT13, WKWO12] und Wiederherstellung [KMR+13, VKR11] nicht gegeben, dort wird auf die rechtzeitige Auslieferung nicht eingegangen.

Zudem wurde in keiner bisherigen Betrachung im Rahmen von CEP-Systemen die Warteschlangenauslastungen der Berechnungsknoten in Verbindung mit dem Parallelisierungsgrad analysiert, um daraufhin Aussagen zum zeitlichen Verhalten zu machen. Dies ist in dieser Arbeit ebenfalls neu.

9 Zusammenfassung und Ausblick

Der parallelisierte und verteilte Architekturansatz für CEP-Systeme ist oft die einzige Möglichkeit, den gestellten Anforderungen gerecht zu werden. Hierzu zählen vor allem die absolute Korrektheit der erkannten Situationen sowie die zeitnahe Auslieferung der komplexen Ereignisse. Kommt es zu Fehlern durch Ausfälle von Rechenknoten, dürfen die zugesicherten Garantien zu keinem Zeitpunkt verletzt werden. Im Rahmen dessen wurde in dieser Diplomarbeit ein Beitrag zur Zuverlässigkeit von CEP-Systemen geleistet. Mithilfe eines mathematischen Modells lassen sich im Voraus die Menge der benötigten Ressourcen berechnen, um auch im Fehlerfall vollständige Korrektheit und Rechtzeitigkeit zu garantieren. Dieses Modell wurde experimentell validiert.

In dieser Arbeit wurden die Konzepte und verschiedenen Herangehensweisen der Parallelisierung und Verteilung erläutert. Die Idee der selektionsbasierten Parallelisierung ist die effiziente Aufteilung des Ereignisstroms in verschiedene Teilströme, um diese parallel nach Korrelationsmustern zu durchsuchen. Für die geschickte Bildung von Selektionen ist Domänenwissen erforderlich. Die parallelisierte Verarbeitung erhöht den Durchsatz am Operator. Um diese Parallelisierung zu ermöglichen, wurde ein eigenes Rahmenwerk entworfen und schließlich umgesetzt. Somit können auch bestehende Operatoren relativ einfach in dieses Rahmenwerk portiert und dort verwendet werden.

Aufgabe war es, anhand des Parallelisierungsrades die Auslastungen der Warteschlangen von Berechnungsknoten zu limitieren, um so rechtzeitig die Selektionen verarbeiten zu können. Die Verarbeitungsgeschwindigkeit der Knoten konnte dabei als ebenso auf die Rechtzeitigkeit einflussnehmende Größe nicht beeinflusst werden. Das System musste auch im Fehlerfall vollständig funktionieren und durfte keine Einschränkungen nach außen hin zeigen. Insbesondere mussten die Korrektheits- und Echtzeitsanforderungen gewahrt bleiben. Zudem sollten zwei unterschiedliche und gegensätzliche Zuverlässigkeitsverfahren zum Einsatz kommen und miteinander verglichen werden.

Zur Lösung des Problems wurde die Warteschlangentheorie herangezogen. Sie ermöglichte die Berechnung pessimistischer Abschätzungen der einzusetzenden Parallelisierungsgrade in den unterschiedlichen Anfangskonfigurationen. Die Überprüfung der berechneten Lösungen erfolgte dabei in mehreren Experimenten. Wie sich durch die Analyse zeigte, kann damit die Zuverlässigkeit von verteilten und parallelisierten Complex-Event-Processing-Systemen insbesondere hinsichtlich Rechtzeitigkeit sowie Korrektheit verbessert werden.

Ausblick

Das Verfahren kann in Zukunft noch verfeinert werden, indem etwa überlappende und unterschiedlich große Selektionen untersucht werden. Auch durch den Einsatz eines schnelleren Fehlerdetektors sollte sich das zeitliche Verhalten bei der Wiederherstellung von Selektionen deutlich optimieren lassen.

In Zukunft kann zudem ein Algorithmus entworfen werden, der die optimale, maximale Warteschlangenauslastung mitsamt einer zu garantierenden Wahrscheinlichkeit berechnet.

Auch das implementierte Rahmenwerk bietet viel Potential für Erweiterungen. So kann es beispielsweise um Verfahren zum automatischen Aufbau der Topologie erweitert werden. Um die Untersuchung von überlappenden Selektionen vollständig zu unterstützen, müssen alle Komponenten daraufhin angepasst werden.

Literaturverzeichnis

[BBJ+08] N. Bansal, R. Bhagwan, N. Jain, Y. Park, D. S. Turaga, C. Venkatramani. Towards Optimal Resource Allocation in Partial-Fault Tolerant Applications. In *INFOCOM*, S. 1319–1327. IEEE, 2008. URL http://dblp.uni-trier.de/db/conf/infocom/infocom2008.html#BansalBJPTV08. (Zitiert auf Seite 77)

[BDWT13] C. Balkesen, N. Dindar, M. Wetter, N. Tatbul. RIP: run-based intra-query parallelism for scalable complex event processing. In *Proceedings of the 7th ACM international conference on Distributed event-based systems*, DEBS '13, S. 3–14. ACM, New York, NY, USA, 2013. doi:10.1145/2488222.2488257. URL http://doi.acm.org/10.1145/2488222.2488257. (Zitiert auf den Seiten 7, 9, 17, 18 und 77)

[BFSF08] A. Brito, C. Fetzer, H. Sturzrehm, P. Felber. Speculative Out-of-order Event Processing with Software Transaction Memory. In *Proceedings of the Second International Conference on Distributed Event-based Systems*, DEBS '08, S. 265–275. ACM, New York, NY, USA, 2008. doi:10.1145/1385989.1386023. URL http://doi.acm.org/10.1145/1385989.1386023. (Zitiert auf Seite 77)

[BKEN09] O. Ben-Kiki, C. C. Evans, I. döt Net. *YAML Ain't Markup Language (YAML™) Version 1.2*, 3. Auflage, 2009. URL http://www.yaml.org/spec/1.2/spec.html. (Zitiert auf Seite 36)

[Bos] S. K. Bose. Queuing Systems/Kendall's Notation, Little's Result, PASTA, M/M/1/∞ Queue. URL http://www.nptel.iitm.ac.in/courses/117103017/5. (Zitiert auf Seite 49)

[Bra11] G. Brandl. *Sphinx 1.1.3 documentation*, 2011. URL http://sphinx-doc.org/. (Zitiert auf Seite 34)

[CJ09] S. Chakravarthy, Q. Jiang. *Stream Data Processing: A Quality of Service Perspective Modeling, Scheduling, Load Shedding, and Complex Event Processing*, S. 48, 196–198. Springer Publishing Company, Incorporated, 1st Auflage, 2009. (Zitiert auf den Seiten 12 und 13)

[CM94] S. Chakravarthy, D. Mishra. Snoop: An Expressive Event Specification Language for Active Databases. *Data Knowl. Eng.*, 14(1):1–26, 1994. doi:10.1016/0169-023X(94)90006-X. URL http://dx.doi.org/10.1016/0169-023X(94)90006-X. (Zitiert auf den Seiten 13, 28 und 29)

[CM12a] G. Cugola, A. Margara. Low Latency Complex Event Processing on Parallel Hardware. *J. Parallel Distrib. Comput.*, 72(2):205–218, 2012. doi:10.1016/j.jpdc.2011.11.002. URL http://dx.doi.org/10.1016/j.jpdc.2011.11.002. (Zitiert auf Seite 15)

[CM12b] G. Cugola, A. Margara. Processing Flows of Information: From Data Stream to Complex Event Processing. *ACM Comput. Surv.*, 44(3):15:1–15:62, 2012. doi:10.1145/2187671.2187677. URL http://doi.acm.org/10.1145/2187671.2187677. (Zitiert auf Seite 13)

[Cro13] D. Crockford. *ECMA-404 – The JSON Data Interchange Format*. Ecma International, 2013. URL http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf. (Zitiert auf Seite 36)

[Dar13] B. Darnell. *Tornado 3.1.1 documentation*. Facebook Inc., 2013. URL http://www.tornadoweb.org/en/stable/index.html. (Zitiert auf Seite 33)

[EB09] M. Eckert, F. Bry. Complex Event Processing (CEP), 2009. URL http://www.gi.de/service/informatiklexikon/detailansicht/article/complex-event-processing-cep.html. (Zitiert auf den Seiten 11 und 12)

[Esp13] EsperTech Inc. Esper – Complex Event Processing, 2013. URL http://esper.codehaus.org/. (Zitiert auf Seite 16)

[GS01] G. Grimmet, D. Stirzaker. *Probability and random processes*, S. 443. Oxford Univ. Press, 2001. (Zitiert auf Seite 51)

[Hir12] M. Hirzel. Partition and Compose: Parallel Complex Event Processing. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, S. 191–200. ACM, New York, NY, USA, 2012. doi:10.1145/2335484.2335506. URL http://doi.acm.org/10.1145/2335484.2335506. (Zitiert auf Seite 77)

[HLR+13] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, B. Koldehofe. Opportunistic Spatio-temporal Event Processing for Mobile Situation Awareness. In *Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems (DEBS)*, S. 195–206. ACM Press, Arlington, Texas, USA, 2013. doi:10.1145/2488222.2488266. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2013-29&engl=. (Zitiert auf Seite 13)

[Int13] International Business Machines Corporation. Website zu WebSphere Business Events, 2013. URL http://www-01.ibm.com/software/integration/wbe/. (Zitiert auf den Seiten 13 und 14)

[JMS+08] N. Jain, S. Mishra, A. Srinivasan, J. Gehrke, J. Widom, H. Balakrishnan, U. Çetintemel, M. Cherniack, R. Tibbetts, S. Zdonik. Towards a Streaming SQL Standard. *Proc. VLDB Endow.*, 1(2):1379–1390, 2008. URL http://dl.acm.org/citation.cfm?id=1454159.1454179. (Zitiert auf Seite 13)

[JSGAW09] G. Jacques-Silva, B. Gedik, H. Andrade, K.-L. Wu. Language level checkpointing support for stream processing applications. In *DSN*, S. 145–154. IEEE, 2009. URL http://dblp.uni-trier.de/db/conf/dsn/dsn2009.html#Jacques-SilvaGAW09. (Zitiert auf Seite 77)

[Ken53] D. G. Kendall. Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain. *The Annals of Mathematical Statistics*, 24(3):338–354, 1953. doi:10.2307/2236285. URL http://projecteuclid.org/DPubS/Repository/1.0/Disseminate?view=body&id=pdf_1&handle=euclid.aoms/1177728975. (Zitiert auf Seite 49)

[KMR+13] B. Koldehofe, R. Mayer, U. Ramachandran, K. Rothermel, M. Völz. Rollback-Recovery without Checkpoints in Distributed Event Processing Systems. In *Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems (DEBS)*. ACM, 2013. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2013-26&engl=. (Zitiert auf den Seiten 10, 33, 42, 45 und 77)

[KN02] G. Klyne, C. Newman. *RFC 3339: Date and Time on the Internet: Timestamps*. Clearswift Corporation and Sun Microsystems, 2002. URL http://www.ietf.org/rfc/rfc3339.txt. (Zitiert auf Seite 25)

[Kol12] B. Koldehofe. Vorlesung Reliable Distributed Programming, 2012. (Zitiert auf Seite 30)

[KORR12] B. Koldehofe, B. Ottenwälder, K. Rothermel, U. Ramachandran. Moving Range Queries in Distributed Complex Event Processing. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems (DEBS).*, S. 201–212. ACM, Berlin, 2012. doi:10.1145/2335484.2335507. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2012-20&engl=. (Zitiert auf Seite 13)

[LS08] D. Luckham, R. Schulte. Event Processing Glossary - Version 1.1. Technischer Bericht, Event Processing Technical Society, 2008. URL http://www.ep-ts.com/component/option,com_docman/task,doc_download/gid,66/Itemid,84/. (Zitiert auf Seite 12)

[Luc02] D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002. (Zitiert auf den Seiten 11 und 31)

[May12] R. Mayer. *Wiederherstellung von Ereignisströmen in CEP-Systemen*. Master's thesis (Diplomarbeit), Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, Germany, 2012. URL ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart_fi/DIP-3336/DIP-3336.pdf. (Zitiert auf den Seiten 9, 12 und 13)

[May13a] R. Mayer. Internal Talk: Reliability and Data Parallelization in Distributed CEP Systems, 2013. Unveröffentlicht. (Zitiert auf den Seiten 7, 9, 50 und 51)

[May13b] R. Mayer. Parallelization of Continuous Complex Event Processing Operators with the ParSe middleware, 2013. Noch unveröffentlicht. (Zitiert auf den Seiten 9, 19, 33 und 34)

[Mis91] D. Mishra. Snoop: An Event Specification Language For Active Database Systems, 1991. URL http://itlab.uta.edu/sharma/PPL/ThesisWeb/deepakm_thesis.pdf. (Zitiert auf den Seiten 11 und 13)

[MMSW07] M. M. Michael, J. E. Moreira, D. Shiloach, R. W. Wisniewski. Scale-up x Scale-out: A Case Study using Nutch/Lucene. In *IPDPS*, S. 1–8. IEEE, 2007. URL http://www.cecs.uci.edu/~papers/ipdps07/pdfs/SMTPS-201-paper-1.pdf. (Zitiert auf Seite 16)

[OKRR13] B. Ottenwälder, B. Koldehofe, K. Rothermel, U. Ramachandran. MigCEP: Operator Migration for Mobility Driven Distributed Complex Event Processing. In *Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems (DEBS)*, S. 183–194. ACM Press, Arlington, Texas, USA, 2013. doi: 10.1145/2488222.2488265. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2013-28&engl=. (Zitiert auf Seite 15)

[Ora13] Oracle Corporation. Website zu Oracle Event Processing, 2013. URL http://www.oracle.com/us/products/middleware/soa/event-processing/overview/index.html. (Zitiert auf den Seiten 13 und 14)

[Pyt13] Python Software Foundation. *Python Programming Language – Official Website*, 2013. URL http://python.org/. (Zitiert auf Seite 33)

[Ray03] E. S. Raymond. *The Art of UNIX Programming*. Pearson Education, 2003. URL http://www.catb.org/esr/writings/taoup/html/ch11s06.html. (Zitiert auf Seite 35)

[RDR10] S. Rizou, F. Dürr, K. Rothermel. Solving the Multi-operator Placement Problem in Large-Scale Operator Networks. In *Proceedings of the 19th International Conference on Computer Communication Networks*, S. 1–6. IEEE Communications Society, Zurich, Switzerland, 2010. doi:10.1109/ICCCN.2010.5560127. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2010-56&engl=. (Zitiert auf Seite 15)

[SKPR10] B. Schilling, B. Koldehofe, U. Pletat, K. Rothermel. Distributed Heterogeneous Event Processing: Enhancing Scalability and Interoperability of CEP in an Industrial Context. In *Proceedings of the 4th ACM International Conference on Distributed Event-Based Systems (DEBS); Cambridge, United Kingdom, July 12-15, 2010*, S. 150–159. ACM, 2010. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2010-22&engl=. (Zitiert auf den Seiten 9 und 13)

[SKR11] B. Schilling, B. Koldehofe, K. Rothermel. Efficient and Distributed Rule Placement in Heavy Constraint-Driven Event Systems. In *Proceedings of the 13th IEEE International Conference on High Performance Computing and Communications (HPCC-2011)*, S. 355–364. IEEE, 2011. doi:10.1109/HPCC.2011.53. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2011-43&engl=. (Zitiert auf Seite 15)

[SKRR13] B. Schilling, B. Koldehofe, K. Rothermel, U. Ramachandran. Access Policy Consolidation for Complex Event Processing. In *IEEE Conference on Networked Systems (NetSys)*. IEEE, 2013. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2013-01&engl=. (Zitiert auf Seite 13)

[SPR09] B. Schilling, U. Pletat, K. Rothermel. Event Correlation in Heterogeneous Environments. *it — Information Technology – Complex Event Processing*, S. 270–275, 2009. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=ART-2009-14&engl=. (Zitiert auf den Seiten 9 und 13)

[Str13] StreamBase Systems, Inc. *StreamSQL Guide*, 2013. URL http://www.streambase.com/developers/docs/latest/streamsql/. (Zitiert auf Seite 13)

[VKR11] M. Völz, B. Koldehofe, K. Rothermel. Supporting Strong Reliability for Distributed Complex Event Processing Systems. In *Proceedings of 13th IEEE International Conference on High Performance Computing and Communications (HPCC-2011)*, S. 477–486. IEEE Computer Society Press, 2011. doi:10.1109/HPCC.2011.69. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2011-44&engl=. (Zitiert auf den Seiten 10, 45 und 77)

[WASW07] A. Widder, R. v. Ammon, P. Schaeffer, C. Wolff. Identification of Suspicious, Unknown Event Patterns in an Event Cloud. In *Proceedings of the 2007 Inaugural International Conference on Distributed Event-based Systems*, DEBS '07, S. 164–170. ACM, New York, NY, USA, 2007. doi:10.1145/1266894.1266926. URL http://doi.acm.org/10.1145/1266894.1266926. (Zitiert auf Seite 13)

[WKWO12] S. Wu, V. Kumar, K.-L. Wu, B. C. Ooi. Parallelizing Stateful Operators in a Distributed Stream Processing System: How, Should You and How Much? In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, S. 278–289. ACM, New York, NY, USA, 2012. doi:10.1145/2335484.2335515. URL http://doi.acm.org/10.1145/2335484.2335515. (Zitiert auf Seite 77)

Alle URLs wurden zuletzt am 28. 11. 2013 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift