Erscheint in: *Ausgezeichnete Informatikdissertationen 2014*
Lecture Notes in Informatics (LNI)
Gesellschaft für Informatik, Bonn 2015

Steffen Hölldobler et al. (Hrsg.): Ausgezeichnete Informatikdissertationen 2014,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2015

Ressourceneffiziente Fehler- und Einbruchstoleranz[1]

Tobias Distler[2]

Abstract: Byzantinisch fehlertolerante Replikation erlaubt es, Systemen die Verfügbarkeit und Zuverlässigkeit von netzwerkbasierten Diensten sogar dann zu garantieren, wenn einige der Replikate, beispielsweise als Folge eines Einbruchs, willkürliches Fehlverhalten zeigen. Obwohl derartige Vorfälle bereits zu schwerwiegenden Dienstausfällen geführt haben, werden existierende Ansätze aus dem Bereich der byzantinischen Fehlertoleranz aufgrund ihres hohen Ressourcenbedarfs weiterhin kaum für den Produktiveinsatz genutzt. Diese Dissertation präsentiert Protokolle und Techniken zur Steigerung der Ressourceneffizienz von byzantinisch fehlertoleranten Systemen. Als Ausgangsbasis dient dabei jeweils die Verwendung zweier Betriebsmodi: Im Normalbetriebsmodus senkt ein System seinen Ressourcenverbrauch so weit, dass Fortschritt nur noch gewährleistet ist, solange sich alle Replikate korrekt verhalten. Im Fehlerbehandlungsmodus stehen dagegen zusätzliche Ressourcen zur Verfügung, um Fehler tolerieren zu können. Ein zentrales Resultat dieser Arbeit ist die Erkenntnis, dass passive Replikation ein effektives Mittel zur Implementierung eines ressourceneffizienten Normalbetriebsmodus darstellt. Darüber hinaus belegen Evaluierungsergebnisse, dass eine verbesserte Ressourceneffizienz auch zu einer Steigerung der Leistungsfähigkeit führen kann.

1 Einleitung

Netzwerkbasierte Dienste werden von ihren Betreibern zunehmend als unentbehrlich angesehen; entweder weil ihr Ausfall direkt zu ökonomischen Verlusten führt, wie etwa bei elektronischen Handelssystemen, oder weil die Verfügbarkeit anderer Dienste von ihnen abhängt, wie es beispielsweise bei Netzwerkdateisystemen [Sun89] oder Koordinierungsdiensten [HKJR10] der Fall ist. Folglich ist es von zentraler Bedeutung, Systeme in die Lage zu versetzen, Fehler in ihren Komponenten tolerieren zu können. Gängige Praxis ist hierbei die Tolerierung von Komponentenausfällen durch Replikation, also die Einführung von Redundanz. Leider zeigt sich jedoch immer wieder, dass die Widerstandsfähigkeit eines Systems gegen Ausfälle allein nicht ausreicht, nämlich genau dann, wenn Störungen auf willkürliches Fehlverhalten zurückzuführen sind [Ama08, App08]. Das Konzept der byzantinisch fehlertoleranten Replikation [LSP82] stellt ein Mittel zur Lösung derartiger Probleme dar, da das zugehörige Fehlermodell explizit beliebiges Fehlverhalten von Systemkomponenten einschließt. Im Besonderen gilt dies auch für Szenarien, in denen ein Angreifer nach einem erfolgreichen Einbruch einen Teil der Replikate kontrolliert und mit ihrer Hilfe aktiv versucht, andere Replikate an der Erfüllung ihrer Aufgaben zu hindern.

Trotz ihrer Vorteile und kontinuierlichen Verbesserungen hinsichtlich Performanz [CL99], Skalierbarkeit [YMV+03] und Implementierungsaufwand [GKQV10] sind byzantinisch fehlertolerante Systeme im Bereich der netzwerkbasierten Dienste aktuell noch nicht im breiten Praxiseinsatz. Als einer der Hauptgründe hierfür wurde der *hohe Ressourcenbedarf*

[1] Englischer Titel der Dissertation: „*Resource-efficient Fault and Intrusion Tolerance*“ [Dis14]

[2] Friedrich–Alexander–Universität Erlangen–Nürnberg (FAU), Department Informatik, Lehrstuhl für Informatik 4: Verteilte Systeme und Betriebssysteme, distler@cs.fau.de

existierender Ansätze identifiziert [KR09]: Im allgemeinen Fall benötigt ein byzantinisch fehlertolerantes System mindestens $3f + 1$ Replikate, um das willkürliche Fehlverhalten von bis zu f dieser Replikate tolerieren zu können. Hinzu kommt der im Vergleich zu ausfalltoleranten Systemen erhöhte Bedarf an Netzwerkressourcen und Rechenkapazität, der für die Ausführung eines byzantinisch fehlertoleranten Einigungsprotokolls anfällt.

Ziel dieser Dissertation ist die Entwicklung neuer Protokolle und Techniken, mit denen sich die Ressourceneffizienz byzantinisch fehlertoleranter Systeme steigern lässt, um so ihren breiten Einsatz in der Praxis zu ermöglichen. Alle im Weiteren präsentierten Ansätze beruhen dabei auf derselben Grundidee: einer klaren Trennung zwischen Normalbetrieb und Fehlerbehandlung. Die fundamentale Erkenntnis hierfür ist, dass es im Normalbetrieb ausreicht, Fehler erkennen (oder sie zumindest vermuten) zu können, wogegen sie im Zuge der Fehlerbehandlung tatsächlich toleriert werden müssen, und dass Ersteres im Allgemeinen weniger Ressourcen erfordert als Letzteres. Traditionelle Herangehensweisen [CL99, YMV+03, VCB+11] sehen dagegen vor, dass byzantinisch fehlertolerante Systeme jederzeit ihren Ressourcenbedarf auf die maximale Anzahl zu tolerierender Fehler ausrichten, sich also quasi in einem Dauerzustand der Fehlerbehandlung befinden.

In der Implementierung eines Systems lässt sich eine Trennung zwischen Normal- und Fehlerfall durch zwei Betriebsmodi realisieren: Befindet es sich im *Normalbetriebsmodus*, reduziert ein System seinen Ressourcenverbrauch, indem es beispielsweise die Anzahl der Replikate verringert, auf denen eine Client-Anfrage redundant ausgeführt wird. Insbesondere ist es einem System in diesem Modus auch erlaubt, auf Protokolle zurückzugreifen, die Ressourcen sparen, indem sie nur Fortschritt garantieren, solange sich alle Beteiligten korrekt verhalten. Da ein System im Normalbetriebsmodus nicht in der Lage ist, beim Auftreten von Fehlern Verfügbarkeit zu gewährleisten, muss jederzeit ein Wechsel in den *Fehlerbehandlungsmodus* möglich sein. In diesem stehen dann widerstandsfähigere Protokolle zur Verfügung, die durch den Einsatz zusätzlicher Ressourcen Fehler tolerieren können, etwa indem sie eine Client-Anfrage von weiteren Replikaten bearbeiten lassen.

Je seltener Fehlersituationen während des Betriebs auftreten, desto länger ist es einem System mit Hilfe des Normalbetriebsmodus möglich, seine Ressourceneffizienz zu steigern. Im Rahmen dieser Dissertation werden, jeweils anhand einer Prototypimplementierung, drei Ansätze für die Realisierung des Normalbetriebsmodus präsentiert und evaluiert:

- SPARE (Abschnitt 3) nutzt die speziellen Eigenschaften virtueller Maschinen, um einen Teil der Replikate in einen ressourcensparenden Zustand zu versetzen. Das System kombiniert erstmals byzantinische Fehlertoleranz mit passiver Replikation.
- REBFT (Abschnitt 4) verallgemeinert den SPARE-Ansatz und ermöglicht so das erste byzantinisch fehlertolerante System, bei dem es zur Einleitung der Fehlerbehandlung ausreicht, wenn eines der im Normalfall aktiven Replikate fehlerfrei ist.
- ODRC (Abschnitt 5) nutzt während des Normalbetriebs frei werdende Ressourcen zur Steigerung der Performanz und widerlegt die allgemeine Ansicht, dass byzantinische Fehlertoleranz notwendigerweise mit Leistungseinbußen verbunden ist.

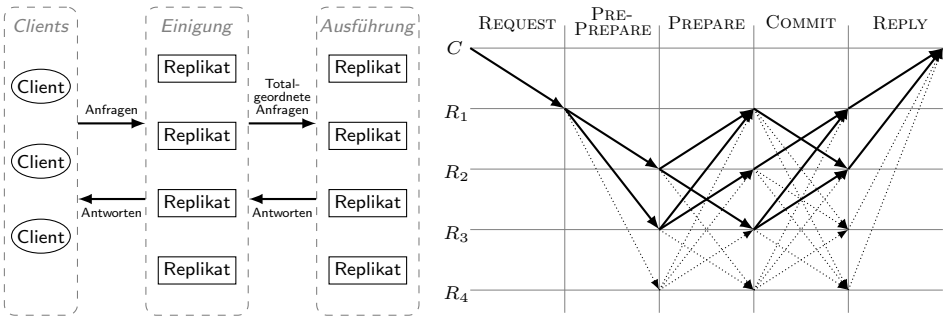
Abschnitt 2 diskutiert zuvor den Ressourcenverbrauch traditioneller Systeme anhand eines Beispiels. Abschnitt 6 fasst schließlich die wichtigsten Ergebnisse der Arbeit zusammen.

2 Systemmodell und Problemanalyse

Im Folgenden wird anhand des für den Bereich der einigungsbasierten byzantinisch fehlertoleranten Systeme bahnbrechenden PBFT [CL99] der grundsätzliche Aufbau solcher Systeme erläutert sowie deren Einsatz von Ressourcen analysiert. Abbildung 1a zeigt, dass sich die Systeme in drei Stufen einteilen lassen: Eine Gruppe von Clients nutzt den angebotenen Dienst durch das Senden von Anfragen. Die Anfragen werden von Replikaten der *Einigungsstufe* [YMV+03] unter Verwendung eines byzantinisch fehlertoleranten Protokolls in eine global eindeutige Reihenfolge gebracht, in der Replikate der *Ausführungsstufe* sie schließlich bearbeiten. Die Unterscheidung zwischen Einigungs- und Ausführungsreplikat dient dabei vor allem der Kennzeichnung der jeweiligen Aufgaben. In den meisten Systemen (darunter PBFT) sind beide Funktionen in einem gemeinsamen Replikat realisiert.

Der Einsatz des Einigungsprotokolls stellt sicher, dass korrekte Ausführungsreplikate alle Anfragen in derselben Reihenfolge bearbeiten und somit einen konsistenten Zustand aufweisen. Dies gilt insbesondere auch dann noch, wenn bis zu f der $3f+1$ Einigungsreplikate fehlerhaft sind. Abbildung 1b zeigt, dass für die Einigung in PBFT mehrere Protokollphasen erforderlich sind: In der PREPREPARE-Phase schlägt das Anführerreplikat R_1 eine vom Client erhaltene Anfrage zur Einigung vor. Als Absicherung gegen einen fehlerhaften Anführer vergewissern sich die anderen Replikate in der PREPARE-Phase anschließend, dass sie über denselben Vorschlag abstimmen. Die Annahme der Anfrage erfolgt schließlich in der COMMIT-Phase. Um Fehler tolerieren zu können, schreitet ein Replikat jeweils erst in die nächste Phase voran, wenn ihm mindestens $2f+1$ übereinstimmende PREPAREs bzw. COMMITs für dieselbe Anfrage vorliegen. Wird diese Anforderung innerhalb einer bestimmten Zeitspanne nicht erfüllt, votiert ein Replikat für die Initiierung eines (hier nicht näher betrachteten) Protokolls zur Ablösung des Anführers.

Nach erfolgreicher Einigung wird eine Anfrage von jedem der Replikate bearbeitet und das Ergebnis als Antwort an den Client gesendet. Dieser akzeptiert das Resultat erst nach Erhalt von $f+1$ identischen Antworten, um gewährleisten zu können, dass mindestens eine dieser Antworten von einem korrekten Replikat stammt und somit richtig ist.



(a) Aufteilung in drei logische Stufen (b) Nachrichtenfluss zwischen Client (*C*) und Replikaten (*R*)

Abb. 1: Überblick über die Architektur und das Einigungsprotokoll von PBFT [CL99] für $f = 1$.

Wie auch andere byzantinisch fehlertolerante Systeme [YMV+03, VCB+11] ist PBFT darauf ausgelegt, zu jeder Zeit die Existenz von f fehlerhaften Replikaten anzunehmen und durch Einsatz entsprechender Redundanzen zu tolerieren. Dies führt dazu, dass das System in Zeiten weniger oder keiner Fehler deutlich mehr Ressourcen verbraucht als eigentlich nötig sind, um unter solchen Bedingungen Fortschritt zu erzielen: Verhalten sich alle Beteiligten korrekt, reichen beispielsweise 13 der 29 gesendeten Nachrichten zum erfolgreichen Abschluss eines Protokolldurchlaufs aus, wie in Abbildung 1b anhand der durchgezogenen Linien illustriert ist. Ebenso tragen in diesem Fall nur zwei (d. h. $f+1$) der vier Ausführungen der Anfrage zur Akzeptanz der Antwort durch den Client bei. Ziel des in dieser Dissertation verfolgten Ansatzes ist es, dieses Potenzial durch die Einführung eines Normalbetriebsmodus zur Steigerung der Ressourceneffizienz zu nutzen. Im Folgenden werden drei Möglichkeiten präsentiert, einen solchen Modus bereitzustellen.

3 Ressourceneffiziente virtualisierungsbasierte Replikation

Der Einsatz von Virtualisierung [BDF+03] erlaubt es Datenzentren, die Dienste verschiedener Nutzer isoliert voneinander auf denselben physischen Rechnern zu betreiben und dadurch Kosten zu sparen. Das hier vorgestellte SPARE[3] zeigt, wie sich mit Hilfe eines auf virtualisierte Umgebungen zugeschnittenen Ansatzes ein ressourceneffizienter Infrastrukturdienst realisieren lässt, der byzantinische Fehlertoleranz für die fehleranfälligsten Teile des Systems bereitstellt: die in virtuellen Maschinen laufenden Anwendungen der Nutzer.

Architektur Wie in Abbildung 2 dargestellt, umfasst SPARE $f+1$ physische Server, auf denen jeweils drei virtuelle Maschinen (VMs) betrieben werden: zwei Nutzer-VMs mit jeweils einem Ausführungsreplikat (inkl. Betriebssystem und Anwendung) und eine privilegierte VM mit der SPARE-Logik, dem *Replikat-Manager*. Alle Replikat-Manager sind über ein privates (partiell synchrones) Netzwerk miteinander verbunden. Clients kommunizieren über ein separates öffentliches (potentiell asynchrones) Netzwerk mit dem Dienst. Da SPARE Nutzer-VMs keinen direkten Netzwerkzugriff gewährt, wird jede ihrer Interaktionen mit der Außenwelt über den lokalen Replikat-Manager abgewickelt.

Fehlermodell Aktuell in Datenzentren genutzte Systeme bieten ausschließlich Schutz gegen Komponentenausfälle. SPARE geht mit seinem hybriden Fehlermodell einen Schritt weiter: Während in dem vom Datenzentrumsbetreiber betreuten Teil der Infrastruktur (der Virtualisierungsschicht, den privilegierten VMs und den Replikat-Managern etc.) ebenfalls nur Ausfälle toleriert werden können, bietet das System dagegen byzantinische Fehlertoleranz für die in Nutzer-VMs ausgeführten Dienstreplikate [RK07]. Eine derartige Unterscheidung fußt auf der Beobachtung, dass die Betreiber von Datenzentren ihre eigenen Systeme regelmäßig warten und ihnen daher vertrauen, wogegen Nutzer in den gemieteten virtuellen Maschinen oftmals vergleichsweise einfach anzugreifenden Code ausführen.

Normalbetrieb Von Clients an den Dienst gestellte Anfragen werden von den jeweiligen Replikat-Managern abgefangen und zunächst mittels eines (ausfalltoleranten) Einigungsprotokolls im privaten Netzwerk geordnet. Anschließend lassen die Replikat-Manager die

[3] *„spare“* (engl.): 1. sparsam; 2. Ersatz, Reserve.

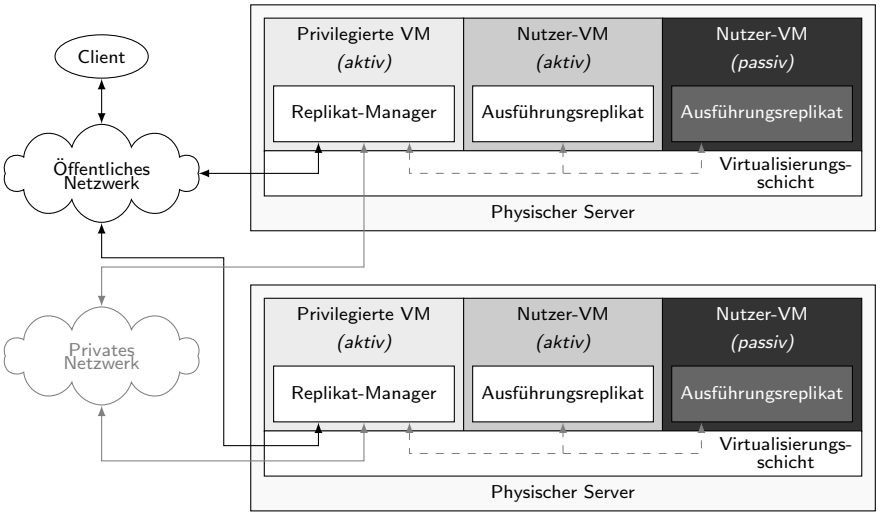


Abb. 2: Einsatz von physischen und virtuellen Maschinen (VMs) in SPARE für $f = 1$.

Anfragen von den Ausführungsreplikaten der lokalen Nutzer-VMs bearbeiten und sammeln deren Antworten. Bevor ein Client sein Ergebnis erhält, wird dessen Korrektheit zunächst von einem Replikat-Manager auf Basis der Antworten unterschiedlicher Replikate verifiziert. Da hierfür in Abwesenheit von Fehlern $f+1$ Antworten ausreichen (siehe Abschnitt 2), nutzt SPARE folgenden Ansatz zur Senkung seines Ressourcenverbrauchs: Solange sich das System im Normalbetriebsmodus befindet, wird jede Anfrage nicht von allen $2f+2$ Dienstreplikaten ausgeführt, sondern nur von $f+1$ *aktiven* (siehe Abbildung 2). Die virtuellen Maschinen der restlichen $f+1$ *passiven* Replikate befinden sich dagegen die meiste Zeit in einem ressourcensparenden Modus, in dem sie nicht läuffähig sind und daher auch nicht vom System eingeplant werden. Um im Fehlerfall dennoch auf sie zurückgreifen zu können, aktualisiert SPARE ihre Zustände in regelmäßigen Abständen. Hierzu wecken die Replikat-Manager die passiven Replikate kurzzeitig auf und versorgen sie mit den neusten Zustandsänderungen. Diese wurden zuvor von den aktiven Replikaten jeweils bei der Bearbeitung modifizierender Anfragen bereitgestellt und von den Replikat-Managern gepuffert sowie (analog zum Verfahren bei Antworten) per Vergleich verifiziert.

Fehlerbehandlung Fehler in aktiven Ausführungsreplikaten können dazu führen, dass die Verifikation von Antworten und/oder Zustandsänderungen aufgrund einer zu geringen Zahl identischer Exemplare stockt. In solchen Fällen wechselt SPARE in den Fehlerbehandlungsmodus und aktiviert einige der passiven Replikate. Wie viele passive Replikate hierbei beteiligt sind, hängt von der Anzahl der voraussichtlich noch zum erfolgreichen Abschluss der Verifikation benötigten Exemplare ab. Im Zuge der Aktivierung sorgen die Replikat-Manager dafür, dass die passiven Replikate zunächst die letzten Zustandsänderungen einspielen, bevor sie die betroffenen Anfragen bearbeiten. Auf diese Weise wird sichergestellt, dass die Replikate zu konsistenten Ergebnissen kommen. Nach Behandlung der Fehlersituation kehrt SPARE in den Normalbetriebsmodus zurück.

Evaluierung SPARE wurde auf Basis der Virtualisierungssoftware Xen [BDF+03] implementiert und mittels RUBiS [RUB], einem Benchmark für Middleware-Systeme, evaluiert. Im Fokus der Experimente stand dabei der Vergleich zu zwei verwandten Ansätzen: CRASH, einem Infrastrukturdienst, der ausschließlich Ausfalltoleranz für in Nutzer-VMs betriebene Anwendungen bietet und APPBFT, einem auf aktiver Replikation basierenden System, das wie SPARE byzantinische Fehlertoleranz für Dienstreplikate bereitstellt, allerdings $2f+1$ physische Server benötigt und über keinen ressourcensparenden Normalbetriebsmodus verfügt [RK07]. Die Ergebnisse der Experimente zeigen, dass sich der Ressourcenverbrauch in SPARE durch den Einsatz von passiven Replikaten signifikant senken lässt: Im Vergleich zu APPBFT benötigt das System 30% weniger Prozessorleistung und sendet 8% weniger Daten über das Netzwerk. SPARE belegt somit kaum mehr Ressourcen (Prozessor: 1%, Netzwerk: 2%) als ein vergleichbares ausfalltolerantes System (CRASH).

4 Passive byzantinisch fehlertolerante Replikation

Anhand von SPARE konnte gezeigt werden, dass passive Replikation eine effektive Methode darstellt, um in virtualisierten Umgebungen Ressourcen auf Ebene der Ausführungsreplikate zu sparen. Im Folgenden wird mit REBFT[4] ein Ansatz präsentiert, der das Prinzip auf die Einigungsstufe erweitert und auch in nicht-virtualisierten Szenarien einsetzbar ist.

Replikate Aktive Replikate beteiligen sich in REBFT sowohl an der Einigung von Anfragen als auch an deren Ausführung. Passive Replikate reduzieren dagegen im Normalbetriebsmodus ihren Ressourcenbedarf, indem sie weder Einigungsnachrichten anderer Replikate erhalten noch eigene senden, und darüber hinaus auch keine Anfragen bearbeiten. Um sie trotzdem bei Bedarf zur Fehlerbehandlung nutzen zu können, werden passive Replikate mittels von aktiven Replikaten stammenden Zustandsänderungen aktuell gehalten.

Moduswechsel Die Anzahl der passiven Replikate ist mit f so gewählt, dass ein System im Normalbetriebsmodus in der Lage ist, Fortschritt zu erzielen, solange alle aktiven Replikate sich korrekt verhalten. Ist dies nicht (mehr) der Fall, schaltet das System in den Fehlerbehandlungsmodus. Neben der Aktivierung der passiven Replikate erfolgt hierbei auch ein Wechsel des Einigungsprotokolls, um ab diesem Zeitpunkt bei der Einigung von Anfragen alle noch im System verfügbaren korrekten Replikate einbeziehen und somit Fehler tolerieren zu können. Eine zentrale Herausforderung besteht in diesem Zusammenhang darin, zu garantieren, dass ein solcher Protokollwechsel selbst dann zuverlässig und konsistent durchgeführt wird, wenn bis zu f der am bisherigen, ressourceneffizienten Einigungsprotokoll beteiligten Replikate fehlerhaft sind, und mitunter sogar versuchen, das Umschalten auf den Fehlerbehandlungsmodus zu verhindern. REBFT stellt dies durch den Einsatz von *Abbruchhistorien* sicher, in denen jedes aktive Replikat zu Beginn der Modusumschaltung seine lokale Sicht auf den Fortschritt des Normalbetriebsprotokolls zum Zeitpunkt des Abbruchs darlegt. Die lokalen Historien verschiedener Replikate werden anschließend zu einer globalen Abbruchhistorie vereint. Diese bildet die Ausgangsbasis für den Start der Fehlerbehandlung und ermöglicht somit einen konsistenten Moduswechsel.

[4] Ressource-efficient Byzantine Fault Tolerance

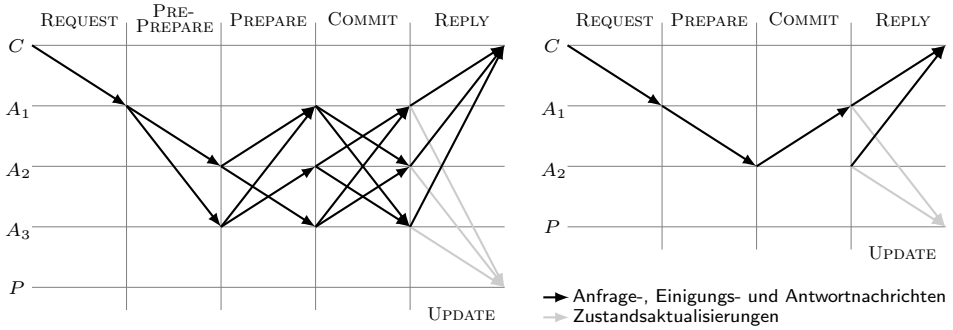


Abb. 3: Nachrichtenfluss während des Normalbetriebs in REPBFT (links) bzw. REMINBFT (rechts) zwischen einem Client (*C*) und den aktiven (*A*) sowie passiven (*P*) Replikaten für $f = 1$.

Anwendung auf existierende Systeme Als Beleg dafür, dass es sich bei REBFT um einen generischen Ansatz handelt, wird das Prinzip in dieser Arbeit in zwei Systemen umgesetzt: PBFT [CL99] und MinBFT [VCB+11]; letzteres benötigt aufgrund der Verwendung von als vertrauenswürdig eingestufter Spezial-Hardware nur $2f + 1$ Replikate zur Tolerierung von bis zu f byzantinischen Fehlern. Die resultierenden Varianten REPBFT und REMINBFT nutzen ihre ursprünglichen Einigungsprotokolle zur Fehlerbehandlung. Die ressourcensparende Einigung während des Normalbetriebs sowie die Modusumschaltung sind dagegen jeweils als zusätzliche Unterprotokolle realisiert [GKQV10]. Dies zeigt, dass sich der REBFT-Ansatz effizient in existierende Implementierungen integrieren lässt. Abbildung 3 präsentiert die Normalbetriebsprotokolle beider Systeme, bei denen die Anzahl der Nachrichten im Vergleich zu den ursprünglichen Einigungsprotokollen signifikant reduziert ist (vergleiche PBFT, Abbildung 1). Trotzdem können sowohl REPBFT als auch REMINBFT aufgrund ihrer auf Abbruchhistorien basierenden (in dieser Zusammenfassung nicht näher beschriebenen) Umschaltprotokolle bei Bedarf einen zuverlässigen Wechsel in den Fehlerbehandlungsmodus garantieren. REMINBFT ist dabei das erste byzantinisch fehlertolerante System, das in der Lage ist, Situationen zu tolerieren, in denen bis auf eines alle im Normalfall aktiven Replikate byzantinisches Fehlverhalten zeigen.

Evaluierung Da der Einfluss passiver Ausführungsreplikate bereits in SPARE analysiert wurde, stand bei der Evaluierung von REBFT das Einigungsprotokoll im Mittelpunkt. Besonders geeignet sind hierfür Experimente, bei denen die Größen für Anfrage- und Antwortnachrichten variiert werden [CL99, GKQV10, VCB+11, YMV+03]. Die Ergebnisse zeigen, dass REPBFT und REMINBFT insbesondere bei großen Anfragen deutliche Vorteile gegenüber PBFT bzw. MinBFT aufweisen: So sinkt etwa für 4 KB große Anfragen die Prozessorlast um 31% bzw. 38% und das über Netzwerk zu übertragende Datenvolumen um 33% bzw. 48%. Im gleichen Zug führt der geringere Mehraufwand während des Normalbetriebs zu einer Steigerung des maximalen Anfragedurchsatzes um 34% bzw. 71%. Weitere Experimente zeigen, dass die Einführung des Normalbetriebsmodus im Fehlerfall keine negativen Folgen nach sich zieht: Der Effekt der Modusumschaltung auf den Systemdurchsatz ist vergleichbar mit dem eines Anführerwechsels in PBFT bzw. MinBFT.

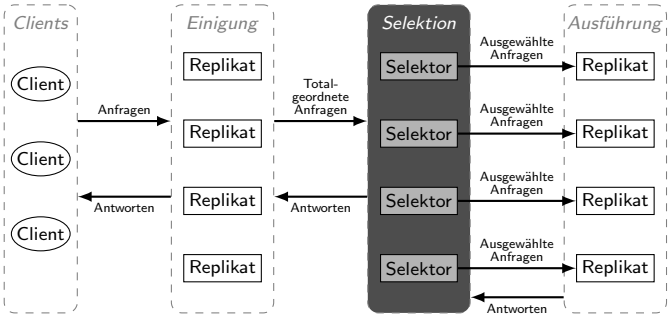


Abb. 4: Einführung einer Selektionsstufe zur Auswahl der je Replikat zu bearbeitenden Anfragen.

5 Bedarfsabhängige Replikatkonsistenz

SPARE und REBFT haben gezeigt, wie sich in einem byzantinisch fehlertoleranten System außerhalb von Fehlerbehandlungen Ressourcen einsparen lassen. Im Unterschied dazu wird in diesem Abschnitt mit ODRC[5] ein Ansatz präsentiert, der die Ressourceneffizienz eines Systems dadurch steigert, dass er während des Normalbetriebs unter Verwendung der vorhandenen Ressourcen die Leistungsfähigkeit des Systems erhöht.

Selektive Ausführung ODRC zielt auf byzantinisch fehlertolerante Systeme ab, deren Struktur die in Abschnitt 2 erläuterten drei logischen Stufen aufweist [CL99, YMV+03, VCB+11]. Wie in Abbildung 4 dargestellt, wird dem bestehenden System hierbei eine zusätzliche Stufe, die *Selektionsstufe*, zwischen Einigung und Ausführung hinzugefügt, die für jedes Ausführungsreplikat jeweils einen eigenen *Selektor* bereitstellt. Dieser entscheidet darüber, welche der geeinigten Anfragen das ihm zugeteilte Replikat bearbeitet und welche nicht. Im Unterschied zu traditionellen Ansätzen wird bei ODRC eine Anfrage nicht von allen Replikaten ausgeführt, sondern nur von einer Untergruppe aus $f+1$ Replikaten. Wie in Abschnitt 2 erläutert, reicht dies während des Normalbetriebs jedoch aus, um die Verifikation der zugehörigen Antwort auf Client-Seite erfolgreich abzuschließen. Da für verschiedene Anfragen unterschiedliche Untergruppen zuständig sind, verteilt sich die Gesamtlast im System. Dies führt dazu, dass bei ODRC bisher belegte Ressourcen frei werden und für die Bearbeitung zusätzlicher Anfragen zur Verfügung stehen.

Normalbetrieb Die Auswahl der jeweils für die Bearbeitung einer Anfrage zuständigen Replikatuntergruppe erfolgt in ODRC auf Basis einer systemweit eindeutigen und jedem Selektor bekannten (anwendungsspezifischen) Abbildungsvorschrift, die jedes Objekt des Anwendungszustands genau $f+1$ Selektoren im System zuordnet. Während des Normalbetriebs hat ein Selektor dafür zu sorgen, dass die ihm zugeordneten Objekte auf seinem Ausführungsreplikat aktuell gehalten werden. Dieser Aufgabe kommt ein Selektor nach, indem er sämtliche Anfragen zur Bearbeitung auswählt, die auf ihm zugeordnete Objekte zugreifen. Alle Anfragen, die diese Anforderung nicht erfüllen, werden dagegen vom Selektor gepuffert und erst bei Bedarf im Rahmen einer Fehlerbehandlung ausgeführt.

[5] On Demand Replica Consistency

Fehlerbehandlung Die Reduzierung der Anzahl redundanter Ausführungen im Normalbetrieb auf $f + 1$ hat zur Folge, dass ein Client bei Auftreten eines Fehlers (zunächst) über zu wenige Antworten verfügt, um das Ergebnis verifizieren zu können. In einem solchen Fall signalisiert der betroffene Client dem System die Ausnahmesituation und leitet dadurch die Fehlerbehandlung ein. Während dieser wählen nun auch die restlichen Selektoren die entsprechende Anfrage zur Bearbeitung aus, so dass der Client weitere Antworten erhält und somit die Verifikation des Resultats abschließen kann. Da ein Selektor im Rahmen von Fehlerbehandlungen Anfragen auswählen muss, die auf ihm nicht zugeordneten Objekten arbeiten, hat er dabei zunächst dafür zu sorgen, dass die Zustände der betroffenen Objekte auf seinem Replikat aktualisiert werden. Hierzu wählt der Selektor zusätzlich all jene der gepufferten Anfragen zur Bearbeitung aus, die auf diese Objekte wirken. Der Aufwand zur Herstellung der Replikatkonsistenz in ODRC ist also abhängig vom konkreten Bedarf der jeweiligen Fehlerbehandlung. Bisherige Ansätze [CL99, YMV+03, VCB+11] halten Replikate dagegen bereits vorsorglich konsistent, indem sie (auch in Abwesenheit von Fehlern) sämtliche Client-Anfragen von allen Replikaten bearbeiten lassen.

Evaluierung Die Evaluierung von ODRC erfolgte auf Basis byzantinisch fehlertoleranter Implementierungen zweier für Datenzentren essentieller Dienste: einem Netzwerkdateisystem (NFS [Sun89]) sowie einem Koordinierungsdienst (ZooKeeper [HKJR10]). In beiden Fällen zeigen die Ergebnisse der Experimente, dass die selektive Ausführung von Anfragen in ODRC während des Normalbetriebs zu signifikanten Leistungssteigerungen führt. So lässt sich im Vergleich zu PBFT beispielsweise ein um 53% höherer Schreibdurchsatz für NFS erzielen. Hervorzuheben ist hierbei insbesondere die Tatsache, dass der von ODRC erzielte Maximaldurchsatz auch um 45% über dem einer nicht-fehlertoleranten Standard-NFS-Implementierung liegt. Dies ist bemerkenswert, da nach allgemeiner Ansicht der Mehraufwand für byzantinische Fehlertoleranz notwendigerweise zu (je nach Anwendungsfall mehr oder weniger starken) Leistungseinbußen führt. ODRC zeigt dagegen, dass sich die zur Tolerierung byzantinischer Fehler erforderlichen Ressourcen während des Normalbetriebs zur Steigerung der Performanz eines Systems nutzen lassen.

6 Zusammenfassung

In traditionellen byzantinisch fehlertoleranten Systemen beteiligen sich zu jedem Zeitpunkt alle korrekten Replikate aktiv an der Einigung und Ausführung von Anfragen, auch wenn dadurch in Abwesenheit von Fehlern mehr Ressourcen verbraucht werden als nötig wären, um Fortschritt zu erzielen. Diese Arbeit hat gezeigt, dass es durch die Einführung eines Normalbetriebsmodus möglich ist, die Ressourceneffizienz solcher Systeme signifikant zu steigern. Hierfür wurden verschiedene Ansätze zur Realisierung eines Normalbetriebsmodus präsentiert und evaluiert, die sich hinsichtlich ihrer Zielsetzung unterscheiden: Der erstmalige Einsatz von passiver Replikation in Kombination mit byzantinischer Fehlertoleranz hat sich als effektives Mittel zur Einsparung von Ressourcen erwiesen. Im Gegensatz dazu erlaubt es die selektive Ausführung von Anfragen, während des Normalbetriebs freie Ressourcen zur Leistungssteigerung zu nutzen. Diese Flexibilität schafft die Voraussetzung für den breiten Praxiseinsatz ressourceneffizienter Systeme, die nicht nur in der Lage sind Komponentenausfälle zu tolerieren, sondern darüber hinaus auch Einbrüche.

Literaturverzeichnis

[Ama08] Amazon S3 Event. `http://status.aws.amazon.com/s3-20080720.html`, 2008.

[App08] Google App Engine Outage. `https://groups.google.com/forum/#!topic/google-appengine/985VmzuLMDs`, 2008.

[BDF+03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt und Andrew Warfield. Xen and the Art of Virtualization. In *Proc. of the 19th Symp. on Operating Systems Principles*, Seiten 164–177, 2003.

[CL99] Miguel Castro und Barbara Liskov. Practical Byzantine Fault Tolerance. In *Proc. of the 3th Symp. on Operating Systems Design and Implementation*, Seiten 173–186, 1999.

[Dis14] Tobias Distler. *Resource-efficient Fault and Intrusion Tolerance*. Dissertation, Friedrich–Alexander–Universität Erlangen–Nürnberg (FAU), 2014.

[GKQV10] Rachid Guerraoui, Nikola Knežević, Vivien Quéma und Marko Vukolić. The Next 700 BFT Protocols. In *Proc. of the 5th European Conf. on Computer Systems*, Seiten 363–376, 2010.

[HKJR10] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira und Benjamin Reed. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In *Proc. of the 2010 USENIX Annual Technical Conf.*, Seiten 145–158, 2010.

[KR09] Petr Kuznetsov und Rodrigo Rodrigues. BFTW3: Why? When? Where? Work. on the Theory and Practice of Byzantine Fault Tolerance. *SIGACT News*, 40(4):82–86, 2009.

[LSP82] Leslie Lamport, Robert Shostak und Marshall Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[RK07] Hans P. Reiser und Rüdiger Kapitza. Hypervisor-based Efficient Proactive Recovery. In *Proc. of the 26th Symp. on Reliable Distributed Systems*, Seiten 83–92, 2007.

[RUB] RUBiS: Rice University Bidding System. `http://rubis.ow2.org/`.

[Sun89] Sun Microsystems. NFS: Network File System Protocol Specification. RFC 1094, 1989.

[VCB+11] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung und Paulo Veríssimo. Efficient Byzantine Fault Tolerance. *IEEE Transactions on Computers*, 62(1):16–30, 2011.

[YMV+03] Jian Yin, Jean-Philippe Martin, Arun Venkataramani, Lorenzo Alvisi und Mike Dahlin. Separating Agreement from Execution for Byzantine Fault Tolerant Services. In *Proc. of the 19th Symp. on Operating Systems Principles*, Seiten 253–267, 2003.

Tobias Distler studierte von 2003 bis 2008 Informatik an der Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU). Seit seinem Abschluss arbeitet er dort als wissenschaftlicher Mitarbeiter bei Prof. Dr.-Ing. Wolfgang Schröder-Preikschat am Lehrstuhl für Verteilte Systeme und Betriebssysteme. Neben einer Lehrtätigkeit in den Bereichen Verteilte Systeme, Middleware und Cloud-Computing umfasst seine Arbeit die Forschung auf den Gebieten Zuverlässigkeit, Sicherheit und Byzantinische Fehlertoleranz. Diese bilden auch den Schwerpunkt seiner Beiträge zu dem durch die DFG geförderten REFIT-Projekt, für die er 2012 einen *IBM Ph.D. Fellowship Award* erhielt. Im Juni 2014 promovierte Tobias Distler mit Auszeichnung an der Technischen Fakultät der FAU.