Effiziente Fehlertoleranz für Web-Basierte Systeme

Marco Serafini

Yahoo! Research
Barcelona, Spain
serafini@yahoo-inc.com

Abstract: Web-Basierte Systeme, die aus einer großen Anzahl von kostengünstigen und unzuverlässigen Rechnern bestehen, werden zunehmend für Dienste benutzt, deren Verfügbarkeit für wirtschaftliche oder persönliche Zwecke kritisch ist. Wir widmen uns in dieser Arbeit der Entwicklung von neuen, effizienten Replikationsalgorithmen für Web-Basierte Systeme, die zwei wichtige Klassen von Fehlern tolerieren. Die erste Klasse fasst Worstcase-Fehler um, die, wie Beispiele aus der nahen Vergangenheit zeigen, bereits den Ausfall wichtiger Online-Dienste verursacht haben. Wir stellen neue Algorithmen vor, die solche Fehler mit niedrigen Replikationskosten und hoher Effizienz tolerieren. Die zweite Klasse bestehet aus Netzwerkpartitionen. Viele Web-Basierte Systeme bieten gemäß dem bekannten CAP-Prinzip nur schwache Konsistenz an um Verfügbarkeit in Gegenwart von diesen Fehlern zu gewärleisten. Das erschwert die Entwiklung zuverlässiger Anwendungen. Wir definieren und implementieren ein neues Konsistenzmodell, genannt *Eventual Linearizability*, welches es ermöglicht Verfügbarkeit zu garantieren ohne die Konsistenz unnötig zu schwächen.

1 Einleitung

Web-basierte Online-Dienste verarbeiten in zunehmendem Maße sensible, personenbezogene oder wirtschaftlich relevante Daten. Die steigende Tendenz, solche Daten in der *Cloud* zu speichern und zu verwalten, erhöht den Bedarf an verlässlichen Realisierungen für eine steigende Anzahl Web-basierter Anwendungen, wie etwa E-Mail, Kalender, Fotoalben oder Online-Banking. Dieser Trend erklärt die zunehmende Verwendung fehlertoleranter Replikationsalgorithmen bei der Implementierung Web-basierter Anwendungen. Die zum Einsatz kommenden Implementierungen reichen von klassischer, stark konsistenter Replikation in Systemen wie Chubby [Bur06] und ZooKeeper [HKJR10] hin zu hochverfügbarer, schwach konsistenter Replikation in Amazons Dynamo [DHJ+07] oder Yahoo!s PNUTS [CRS+08].

Wir stellen neuartige Algorithmen für fehlertolerante Replikation vor, mit dem Ziel, die Effizienz, Verfügbarkeit und Wirtschaftlichkeit dieser Mechanismen zu erhöhen. Wenngleich die vorgestellten Algorithmen allgemein anwendbar sind, erfüllen sie zwei Eigenschaften, die wesentlich durch den Einsatz in Web-basierten Systemen motiviert sind. Die erste Eigenschaft ist das Tolerieren von Worstcase-Fehlern, in der Literatur auch als *Byzantine* [LSP82] bezeichnet, um eine zuverlässige Verarbeitung sensibler Daten zu gewährleisten. Die zweite Eigenschaft ist die Bereitstellung einer geeigneten Semantik

schwacher Konsistenz für Systeme, für die höchstmögliche Verfügbarkeit im Gegenwart von Netzwerkpartitionen zusammen mit geringstmöglicher Zusatzaufwand hinsichtlich Performanz und Replikation sichergestellt werden soll, Abschwächungen der Konsistenz aber weitgehend zu vermeiden sind.

Dieses Dokument ist gegliedert wie folgt. In Sektion 2 diskutieren wir effiziente Algorithmen, um *Byzantine*-Fehler zu tolerieren. Zunächst stellen wir in Sektion 2.1 Scrooge vor und zeigen damit, dass zusätzliche Replikationskosten zur Erzielung einer niedrigeren Latenz und eines höheren Durchsatzes an Operationen ausschließlich von der Anzahl der zu tolerierenden *Byzantine*-Fehlern abhängt. Dieses Ergebnis ist interessant für Web-basierte Systeme, in denen Ausfälle häufig, *Byzantine*-Fehler dagegen selten sind. In Sektion 2.2 führen wir ein neues *Fail-Heterogeneous* Architekturmodell, das auch vertrauenswürdige Komponenten erlaubt. Wir beschreiben einen Algorithmus, genannt HeterTrust, der die These frühere Arbeiten widerlegt, und zeigt dass die Annahme ein synchrones Netzwerk nicht nötig ist, um die Effizienzvorteile vertrauenwürdiger Komponenten ausnutzen zu können. In Sektion 3 definieren wir eine neue Eigenschaft, genannt *Eventual Linearizability*, die die Abschwächung von starke Konsistenz (*Linearizability*) für endliche Zeitfenster erlaubt. Wir stellen einen Algorithmus vor, genannt Aurora, der diese Abschwächung nur dann zulässt, wenn es notwendig ist Verfügbarkeit zu garantieren. Wir zeigen, dass bei der Kombination von *Eventual Linearizability* und *Linearizability* ein inhärenter Aufwand exisitert, da ein stärker Fehlerdetektor gebraucht benötigt wird als für Replikationsalgorithmen die nur *Linearizability* implementieren.

2 Toleranz von *Byzantine*-Fehlern

Das Tolerieren von *Byzantine*-Fehlern (englisch: *Byzantine Fault Tolerance*, BFT) ist derzeit Gegenstand intensiver Forschung. Dabei besteht das Hauptforschungsziel darin, den von BFT implizierten Zusatzaufwand (bzgl. Performanz und erforderlicher Replikation) soweit zu reduzieren, das er mit dem herkömmlicher Fehlertoleranzmechanismen vergleichbar ist. BFT wird zumeist durch die Replikation von Zustandsautomaten erzielt, indem für den Nutzer die Illusion eines einzelnen, zuverlässigen Servers durch die (transparente) Koordination mehrerer unzuverlässiger Server erzeugt wird [Sch90]. Die Anwendbarkeit dieses Ansatzes auf Web-basierte Systeme ist als das endgültiges Ziel dieser Forschungsrichtung zu sehen [CKL+09]. Zweck ist es die so implementierten, kritischen Anwendungen vor folgenschwerem Fehlverhalten (wie etwa in [Das]) zu schützen. Wir stellen neue Algorithmen vor, die den Performanz- und Replikationsaufwand von BFT reduzieren.

2.1 Schnelle BFT mit niedrigen Replikationskosten

Zunächst stellen wir Algorithmen vor, die keine Annahmen bezüglich vertrauenswürdiger Komponenten benötigen [SBD+10]. Nach der Veröffentlichung des richtungsweisenden

PBFT-Algorithmus [CL99] wurde eine Reihe *schneller* BFT-Algorithmen, wie zum Beispiel [MA06, DGV04, KAD+07] entwickelt. Die wichtigsten Protokolle sind in der Tabelle 1 aufgelistet. Ein Protokoll wird in Tabelle 1 als *schnell* bezeichnet, wenn es die minimale Latenzzeit, in der das *Consensus*-Problem gelöst [MA06, DGV04] werden kann, erreicht.

	Replikationskosten (Min. $2f + b + 1$ [Lam03])	*Schnell* ohne unerreichbare Replikas	*Schnell* mit f unerreichbaren Replikas
PBFT [CL99]	$3f + 1$	NO	NO
Zyzzyva [KAD+07]	$3f + 1$	YES	NO
Zyzzyva5 [KAD+07]	$5f + 1$	YES	YES
DGV [DGV04]	$3f + 2b - 1$	YES	YES
Scrooge [SBD+10]	$2f + 2b$	YES	YES

Tabelle 1: Vergleich von *primary-based* BFT-Replikationsprotokollen, die f Fehler, inklusiv $b \leq f$ *Byzantine*-Fehler, tolerieren. Wenn nur f angegeben ist, dann ist $b = f$. DGV hat Replikationskosten von $2f + 2b + 1$ wenn $f = 1$.

Die oben genannten Arbeiten zeigen unter der Annahme fehlerbehafteter Replikas einen inhärenten Konflikt zwischen optimaler Redundanz und minimaler Latenzzeit. In Web-basierten Systemen, in denen *Byzantine*-Fehler nur selten, Ausfälle von Replikas hingegen häufig auftreten, stellt sich dieser unvermeidbare Konflikt als problematisch heraus.

Wir stellen den Scrooge-Algorithmus vor, der den Replikationsaufwand schneller BFT-Replikation in Gegenwart nicht mehr reagierender Replikas reduziert. Dazu wird unmittelbar nach dem Ausfallen von Replikas eine Rekonfigurationsprozedur durchgeführt, die nur von kurzer Dauer und in der Praxis vernachlässigbar ist, da dadurch die replizierte Anwendung nicht unverfügbar sondern nur kurzzeitig und wenig langsamer wird. Um die Replikationskosten weiter zu reduzieren, nutzt Scrooge eine Eigenschaft der üblichen *Message Authentication Codes* aus, die von vergleichbaren Algorithmen nur zur Authentifizierung von Nachrichten benutzt werden. Scrooge benötigt dazu keine zusätzlichen kyptographischen Operationen als andere Algorithmen.

Ein wichtiger, konzeptueller Beitrag von Scrooge ist, dass der zusätzliche Replikationsaufwand zum Erreichen einer höheren Geschwindigkeit ausschließlich von der Anzahl der zu tolerierenden, fehlerbehafteten Replikas (b) abhängt und nicht von der Anzahl der zu tolerierenden Ausfälle ($f - b$). In Konsequenz erzielt Scrooge optimale Robustheit im Fall eines einzelnen *Byzantine*-Fehlers und einer beliebigen Anzahl von Ausfällen. Ein solches Szenario ist charakteristisch für Web-basierte Systeme, in denen *Byzantine*-Fehler äußerst selten auftreten.

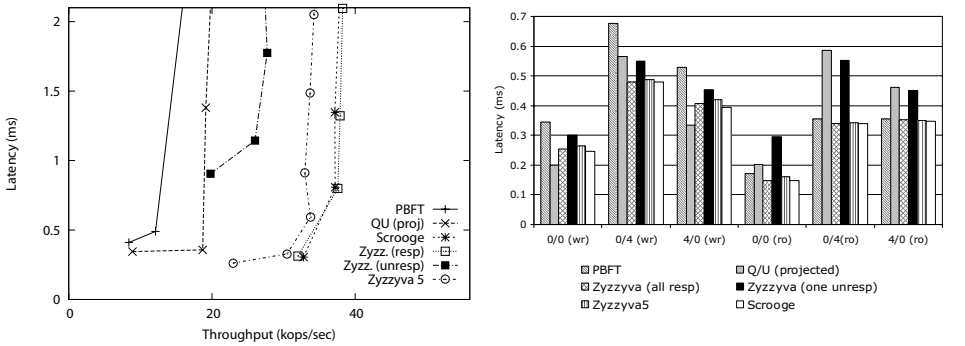
Wir haben Scrooge implementiert und mit PBFT und Zyzzyva verglichen. In Durchläufen mit einer unerreichbaren Replika haben wir die folgende Ergebnisse gefunden (siehe Abbildung 1).

- Der maximale Durchsatz an Operationen von Scrooge ist mehr als 1,3-fach über dem von Zyzzyva. Scrooge gewährt somit niedrigere Latenzzeit bei hoher Belastung (siehe Abbildung 1(a));
- Scrooge reduziert die Latenzzeit bei niedriger Belastung um 20 bis 80 Prozent im

Vergleich zu Zyzzyva (siehe Abbildung 1(b));

- Scrooge hat eine vergleichbare Leistung wie Zyzzyva5, das aber $f+1$ mehr Replikas als Scrooge benötigt;

In fehlerfrei Durchläufen zeigen Scrooge und Zyzzyva die selbe Leistungen.



(a) Durchsatz an Operationen und Latenzzeit

(b) Latenzzeit einzelner *write-read* und *read-only* Operationen (*wr* bzw. *ro*). PBFT, Zyzzyva und Scrooge bieten Optimierungen für *read-only* Operationen, die den Zustand nicht modifizieren, an.

Abbildung 1: Experimentelle Evaluierung von Scrooge im Vergleich mit anderen BFT Algorithmen. Die wichtigsten leistungsrelevanten Eigenschaften von PBFT und Zyzzyva sind in der Tabelle 1 zusammengefasst. Für Zyzzyva zeigen die Abbildungen zwei verschiedene Experimente: ohne und mit unerreichbaren Replikas (*resp* bzw. *unresp*). Als Referenz zeigen wir auch die Ergebnisse von Q/U, ein *Quorum-based* Replikationsalgorithmus [AEMGG+05].

2.2 Integration von vertrauenswürdigen Komponenten

Im folgenden diskutieren wir nun die potenziellen Vorteile in der Verwendung vertrauenswürdiger Komponenten, die nur ausfallen können. Wir zeigen, dass diese zu einer signifikanten Reduktion der Latenz und der durch die Redundanz verursachten Kosten in anwendungstypischen, asynchronen Systemen führen können [SS07]. Dies widerlegt die These früherer Arbeiten [CNV04, Ver06], dass eine Kostenreduktion durch vertrauenswürdige Komponenten zwingend die Verfügbarkeit synchroner Kommunikationskanäle erfordert. Diese zusätzliche Forderung nach Synchronität beschränkt bestehende Lösungen deutlich in möglichen Einsatzgebieten wie beispielsweise in Web-basierten Systemen. In dreistufig organisierten Web-basierten Systemen, kann man sich die Eigenschaft zunutze machen, dass Server in der ersten Ebene des Systems (die Webserver) sehr oft standardisiert, stabiler und weniger fehleranfällig sind als beispielsweise Application-Servers, die in der Regel ad-hoc Anwendungen ausführen.

Wir schlagen ein neues *Fail-Heterogeneous* Architekturmodell für verteilte Systeme vor, das zwei Klassen von Knoten erlaubt: vollständige *Execution Servers*, die *Byzantine*-Fehler haben können, und einfachere, vertrauenswürdige *Dedicated Coordinators*, die nur ausfallen können. *Dedicated Coordinators* sollen generell und einfach sein, um die wahrscheinlichkeit von Fehlern bei der Entwicklung zu minimieren. Wir führen ein neues BFT Protokoll ein, genannt HeterTrust, um die Vorteile zu zeigen, die unser Architekturmodell für die Toleranz von *Byzantine*-Fehlern ermöglicht.

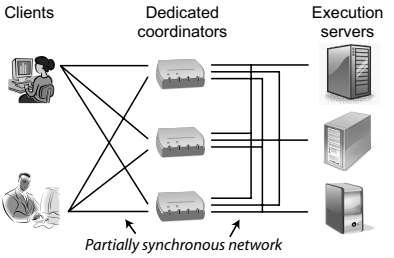


Abbildung 2: Das *Fail-Heterogeneous* Architekturmodell - ein Beispiel.

Das HeterTrust Protokoll, benötigt nur eine, zur Anzahl der zu tolerierenden Fehler lineare, Anzahl von Replikas um die Vertraulichkeit von Daten sicherzustellen und besitzt konstante Komplexität. Dies stellt eine deutliche Verbesserung gegenüber bestehenden Ansätzen dar (siehe Tabelle 2), die zwar keine vertrauenswürdigen Komponenten voraussetzen, aber quadratische Redundanzkosten und lineare Latenzzeit mit sich bringen [YMV$^+$03]. Weiterhin, im Gegensatz zu anderen aus Vertraulichkeit basierenden BFT-Ansätzen, verwendet HeterTrust symmetrische Kryptografie anstelle von Public-Key-Verfahren.

Protokoll	**SM**	**FM**	***n***	**Latenzzeit**	**Geheimhaltung**	**Kryptographie**
Paxos [Lam98, Lam01]	PS	C	$2g+1$	$4/5$	-	-
BFT [CL99]	PS	B	$3f+1$	4	nein	MAC
FaB [MA06]	PS	B	$5f+1$	3	nein	MAC
Correia *et.al* [CNV04]	W	W	$2m+1$	5	nein	MAC
Marchetti *et.al* [MBTPV06]	PS	C	$3g+2$	$4/5$	-	-
Yin *et.al* [YMV$^+$03]	PS	B	f^2+6f+2	$2f+7$	ja	TS
HeterTrust [SS07]	PS	H	$2f+2g+2$	4	ja	MAC

n = untere Schranke für die Anzahl von Replikas
$g/f/m$ = obere Schranke für die Anzahl von *Crash / Byzantine / mixed* Fehlern
SM = Systemmodell (***P**artially **S**ynchronous / **W**ormhole)*
FM = Fehlermodell (***C**rash / **B**yzantine / **W**ormhole / **H**eterogeneous)*
MAC = *Message Authentication Codes*; TS = *Threshold Signatures*

Tabelle 2: Vergleich zwischen HeterTrust und andere BFT Protokollen

Unsere Arbeit umfasst, u.a., die folgenden Beiträge:

- Wir definieren und führen das *Fail-Heterogeneous* Architekturmodell ein, benutzen Replikation als Fallstudie, und stellen den HeterTrust-Algorithmus vor;
- Wenn jede Replika Zugriff auf eine vertrauenswürdige Komponente hat, dann zeigen wir, dass die minimale Anzahl von Replikas um f *Byzantine*-Fehler zu tolerie-

ren, von $3f + 1$ auf $2f + 1$ reduziert werden kann, ohne dass die Annahme eines synchronen Netzwerks erforderlich ist;

- Wir zeigen, dass es mit HeterTrust möglich ist, die Vertraulichkeit der Daten ohne die Verwendung asymmetrischer Kryptographie zu garantieren, wenn $2g + 1$ vertrauliche Komponenten zu Verfügung stehen und g davon ausfallen können;
- Wir führen Techniken für ein um *Denial of Service* Angriffe zu tolerieren.

Das HeterTrust-Protokoll verwendet Ideen in den Bereichen Spekulation [KAD+07] und der Toleranz von *Denial of Service* Angriffen [ACKL08, CWA+09], deren Eigenschaften in weiteren Arbeiten untersucht und in unmittelbarer Folge von [SS07] publiziert wurden. Gleichzeitig zu der Bearbeitung der vorliegenden Arbeit wurde die Verwendung vertrauenswürdiger Komponenten in asynchronen Systemen unabhängig in [CMSK07] untersucht.

3 Toleranz von Netzwerkpartitionen

Viele Web-Basierte Systeme sind als sogenannte konkurrente Systeme entwickelt, wobei eine Menge von sequentiellen Prozessen über *Shared Objects* miteinander kommuniziert. Fehlertoleranz wird dadurch erreicht, dass jeder Prozess eine lokale Kopie des *Shared Object* speichert. Die Prozesse müssen ihre Operationen auf den lokalen Kopien koordinieren, um die gewünschte Konsistenz zu erreichen. Je stärker diese Konsistenz ist, umso einfacher gelingt es, Anwendungen mit Hilfe der *Shared Objects* zu entwickeln.

Eine starke Konsistenzeigenschaft ist Linearisierberkeit (*Linearizability*), die den Prozessen die Illusion bietet, über ein einzelnes, zentralisiertes *Shared Object* zu kommunizieren [HW90]. *Linearizability* kann jedoch nur unter denselben Voraussetzungen implementiert werden, unter denen auch das *Consensus*-Problem lösbar ist. In asynchronen Systemen, wo Prozesse mittels Versenden von Nachrichten kommunizieren, kann *Consensus* nur gelöst werden, wenn ein Fehlerdetektor der Klasse $\diamond\mathcal{S}$ oder der äquivalenten *Leader Oracle* Klasse Ω und eine Mehrheit korrekter Prozesse verfügbar sind [CT96, CHT96].

Für einige Web-basierte Anwendungen können starke Konsistenzeigenschaften wie *Linearizability* nicht erreicht werden, da Verfügbarkeit der Anwendung auch in Abwesenheit der o.g. beiden Voraussetzungen gewährleistet werden soll. Diese Unmöglichkeit, *linearizable Shared Objects*, wie zum Beispiel Datenbanken, in Fall von Partitionierung oder Asynchronität zu implementieren, ist auch als CAP-Prinzip (*Consistency, Availability, Partition-tolerance: take two*) bekannt [GL02]. Allerdings geben, bis auf wenige Ausnahmen, alle bekannten Systeme Linearisierbarkeit auch in Zeitintervallen auf, in denen weder Partitionierung noch Asynchronität vorliegen. Da solch eine Schwächung der Konsistenzeigenschaft für einige Anwendungen problematisch ist, konzentrieren sich neuere Forschungsergebnisse auf stärkere Konsistenzeigenschaften, die sich mit Hochverfügbarkeit kombinieren lassen (z.B. [CRS+08]).

Wir stellen *Eventual Linearizability* als neue Konsistenzeigenschaft vor, die eine Verlet-

zung von Linearisierbarkeit für eine endliche Zeitspanne erlaubt [SDM+10]. *Eventual Linearizability* ist die stärkste bekannte Konsistenzeigenschaft, die nur mit $\diamond\mathcal{S}$ implementiert werden kann, unabhängig von der Anzahl korrekter Prozesse. Wir zeigen dass *Eventual Consistency*, eine weitverbreitete Konsistenzeigenschaft, schwächer als *Eventual Linearizability* ist, weil *Linearizability* beliebig oft Verletzen kann.

Einen weiteren Beitrag bildet Aurora, ein Algorithmus, der Linearisierbarkeit in Phasen in denen ein einzelner Leader im System vorhanden ist, sicherstellt. Die von Aurora bereitgestellten Eigenschaften verringern sich graduell mit sich verschlechternden Rahmenbedingungen. Aurora verwendet einen einzelnen, a priori nicht näher bestimmten Fehlerdetektor, von dessen Stärke jedoch die Eigenschaften von Aurora abhängen, z.B. erfordert *Eventual Linearizability* einen $\diamond\mathcal{S}$ Fehlerdetektor. Nur in Phasen von totaler Asynchronität, in denen das rechtzeitige Empfangen von Nachrichten und die Existenz eines einzelnen Leaders nicht gewährleistet und deswegen *Consensus* nicht gelöst werden kann, reduzieren sich die von Aurora garantierten Eigenschaften auf *Eventual Consistency* [FGL+96, Vog09] und *Causal Consistency* [Lam78]. Für diese Eigenschaften erfordert Aurora lediglich einen Fehlerdetektor $\mathcal{C}$ der die *Strongly Complete*-Eigenschaft erfüllt, welche jedoch von jedem bekannten Fehlerdetektor implementiert wird.

Wir untersuchen zusätzlich, ob die Realisierung sogenannter *Strong Operations*, die *Linearizability* erfordern, und *Weak Operations*, für die *Eventual Linearizability* ausreicht, kombiniert werden kann. Aurora verwendet einen $\diamond\mathcal{P}$ Fehlerdetektor damit *Strong Operations* immer terminieren können. Dieser Fehlerdetektor ist stärker als $\diamond\mathcal{S}$, was ausreichend ist, um *linearizable Shared Objects* zu implementieren, wobei alle Operationen *Strong Operations* sind. Diese Beobachtung legt nahe, dass eine inhärente Einschränkung bei der Kombination von *Eventual Linearizability* und *Linearizability* bestehen könnte. Wir haben formal bewiesen, dass eine solche Einschränkung existiert, und $\diamond S$ nicht ausreicht um die Terminierung von *Strong Operations* zu garantieren, wenn sie konkurrent zu *Weak Operations* sind.

4 Conclusions

Wir haben Replikationsalgorithmen vorgestellt, die zwei wichtige Klassen von Fehlern tolerieren: *Byzantine*-Fehler und Netzwerkpartitionen. Die Toleranz von Fehlern dieser beiden Klassen wird zunehmend wichtiger für Web-basierte Systeme, und das hat unsere Forschungsrichtung motiviert. Unsere Beiträge behandeln Theorie und Praxis von zuverlässigen verteilten Systeme, eine Kombination, die immer mehr in diesem heißen Forschunsgebiet benötigt wird. In der Tat, unsere aktuellen Projekte bei Yahoo! Research handeln sich genau um die Entwicklung von innovativen fehlertoleranten Algorithmen, wie z.B. Zab [JRS11], die schon täglich *in production* eingesetzt werden.

Literatur

[ACKL08] Yair Amir, Brian Coan, Jonathan Kirsch und John Lane. Byzantine replication under attack. In *DSN '08: Proceedings of the 38th IEEE/IFIP International Conference on Dependable Systems and Networks*, Seiten 105–114, 2008.

[AEMGG+05] Michael Abd-El-Malek, Gregory R. Ganger, Garth R. Goodson, Michael K. Reiter und Jay J. Wylie. Fault-scalable Byzantine fault-tolerant services. *SIGOPS Operating Systems Review*, 39(5):59–74, 2005.

[Bur06] Mike Burrows. The Chubby lock service for loosely-coupled distributed systems. In *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*, Seiten 335–350, 2006.

[CHT96] Tushar Deepak Chandra, Vassos Hadzilacos und Sam Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, 1996.

[CKL+09] Allen Clement, Manos Kapritsos, Sangmin Lee, Yang Wang, Lorenzo Alvisi, Mike Dahlin und Taylor Riche. Upright cluster services. In *SOSP '09: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, Seiten 277–290, 2009.

[CL99] Miguel Castro und Barbara Liskov. Practical Byzantine Fault Tolerance. In *OSDI '09: Proceedings of the third symposium on Operating Systems Design and Implementation*, Seiten 173–186, 1999.

[CMSK07] Byung-Gon Chun, Petros Maniatis, Scott Shenker und John Kubiatowicz. Attested append-only memory: making adversaries stick to their word. *SIGOPS Operating Systems Review*, 41(6):189–204, 2007.

[CNV04] Miguel Correia, Nuno Ferreira Neves und Paulo Verissimo. How to Tolerate Half Less One Byzantine Nodes in Practical Distributed Systems. In *SRDS '04: Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems*, Seiten 174–183, 2004.

[CRS+08] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver und Ramana Yerneni. PNUTS: Yahoo!'s hosted data serving platform. In *VLDB '08: Very Large Data Bases Conference*, 2008.

[CT96] Tushar Deepak Chandra und Sam Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2):225–267, March 1996.

[CWA+09] Allen Clement, Edmund Wong, Lorenzo Alvisi, Mike Dahlin und Mirco Marchetti. Making Byzantine fault tolerant systems tolerate Byzantine faults. In *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, Seiten 153–168, 2009.

[Das] Amazon Web Services Service Health Dashboard. Amazon S3 Availability Event: July 20, 2008. http://status.aws.amazon.com/s3-20080720.html.

[DGV04] Partha Dutta, Rachid Guerraoui und Marko Vukolić. Best-Case Complexity of Asynchronous Byzantine Consensus. Bericht LPD-REPORT-2008-08, EPFL, 2004.

[DHJ+07] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall und Werner Vogels. Dynamo: Amazon's Highly Available Key-Value Store. In *SOSP '07: ACM Symposium on Operating Systems Principles*, Seiten 205–220, 2007.

[FGL+96] Alan Fekete, David Gupta, Victor Luchangco, Nancy Lynch und Alex Shvartsman. Eventually-serializable data services. In *PODC '96: Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, Seiten 300–309, 1996.

[GL02] Seth Gilbert und Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002.

[HKJR10] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira und Ben Reed. ZooKeeper: Wait-free coordination for Internet-scale systems. In *ATC '10: Proceedings of USENIX Annual Technical Conference*, 2010.

[HW90] Maurice P. Herlihy und Jeannette M. Wing. Linearizability: a correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463–492, 1990.

[JRS11] Flavio Junqueira, Benjamin Reed und Marco Serafini. Zab: High-performance broadcast for primary-backup systems. In *DSN '11: Proceedings of the IEEE International Conference on Dependable Systems and Networks*, 2011.

[KAD+07] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement und Edmund Wong. Zyzzyva: Speculative Byzantine Fault Tolerance. In *SOSP '07: ACM Symposium on Operating Systems Principles*, Seiten 45–58, 2007.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of ACM*, 21(7):558–565, 1978.

[Lam98] Leslie Lamport. The part-time parliament. *ACM Transactions on Computing Systems*, 16(2):133–169, 1998.

[Lam01] Leslie Lamport. Paxos Made Simple. *ACM SIGACT News*, 32(4):18–25, December 2001.

[Lam03] Leslie Lamport. Lower Bounds for Asynchronous Consensus. In *FuDiCo '03: Proceedings of the Future Directions in Distributed Computing Workshop*, Seiten 22–23, October 2003.

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

[MA06] Jean-Philippe Martin und Lorenzo Alvisi. Fast Byzantine Consensus. *IEEE Transactions on Dependable and Secure Computing*, 3(3):202–215, July 2006.

[MBTPV06] Carlo Marchetti, Roberto Baldoni, Sara Tucci-Piergiovanni und Antonino Virgillito. Fully Distributed Three-Tier Active Software Replication. *IEEE Transactions on Parallel and Distributed Systems*, 17(7):633–645, 2006.

[SBD+10] Marco Serafini, Peter Bokor, Dan Dobre, Matthias Majuntke und Neeraj Suri. Scrooge: Reducing the Costs of Fast Byzantine Replication in Presence of Unresponsive Replicas. In *DSN '10: Proceedings of the 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2010.

[Sch90] Fred B. Schneider. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Computing Surveys*, 22(4):299–319, 1990.

[SDM+10] Marco Serafini, Dan Dobre, Matthias Majuntke, Péter Bokor und Neeraj Suri. Eventually linearizable shared objects. In *PODC '10: Proceeding of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, Seiten 95–104, 2010.

[SS07] Marco Serafini und Neeraj Suri. The Fail-Heterogeneous Architectural Model. In *SRDS '07: Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems*, Seiten 103–113, 2007.

[Ver06] Paulo Verissimo. Travelling through Wormholes: a New Look at Distributed Systems Models. *ACM SIGACT News*, 37(1):66–81, 2006.

[Vog09] Werner Vogels. Eventually consistent. *Communications of ACM*, 52(1):40–44, 2009.

[YMV+03] Jian Yin, Jean-Philippe Martin, Arun Venkataramani, Lorenzo Alvisi und Mike Dahlin. Separating agreement from execution for byzantine fault tolerant services. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, Seiten 253–267, 2003.

Marco Serafini besitzt einen Laurea Abschluss mit Auszeichnung von der Universität von Florenz, Italien, und einen Doktortitel mit Auszeichnung von der Technische Universität Darmstadt, Deutschland. Er arbeitet jetzt bei Yahoo! Research. Sein Forschungsinteresse umfasst zuverlässige, verteilte Systeme und große Datenverarbeitungssysteme. Seine Arbeiten wurden, u.a., in der Zeitschrift IEEE TDSC und in den Konferenbänden von ACM PODC und IEEE DSN veröffentlicht. Er arbeitete an vier EU-Projekten mit und erhielt ein GK-Stipendium von der Deutschen Forschungsgemeinschaft. In Kollaboration mit Hitachi Research ist er Erfinder eines Patentes über Membership-Protokolle für synchrone verteilte Systeme. Er ist oder war Gutachter für Zeitschriften wie IEEE TOC, IEEE Computer, ACM Computing Surveys, Distributed Computing Journal, und für Konferenzen wie IEEE DSN und ACM PODC. Er ist oder war Mitglied der Program-Committees für EDCC 2012, SOSFEM 2011, EWDC 2011, und WRAITS 2011.