Zuverlässiger, transaktional verteilter Speicher

Stefan Frenz

Robert Bosch GmbH
stefan.frenz@googlemail.com

Abstract: Beim Betrieb von Rechnern sind Ausfälle von Hardware, Fehler in der Software sowie nicht reproduzierbare Störungen in der Kommunikation zu erwarten. Stützt sich die Funktionsfähigkeit eines aus vielen Rechnern bestehenden Gesamtsystems auf die Verfügbarkeit aller Teilkomponenten, ist das Ausfallrisiko deutlich erhöht. Um die Zuverlässigkeit dennoch ausreichend gewährleisten zu können, ist es wünschenswert, auftretende Fehler zu erkennen und angemessen darauf zu reagieren. Die meisten Verfahren zur Behandlung von Fehlern sind auch im fehlerfreien Betrieb mit zum Teil erheblichen Kosten verbunden und nur auf eine sehr kleine Menge von Fehlern anwendbar. Hieraus ergibt sich der Bedarf einer unter Umständen nicht völlig verlustfreien, dafür aber möglichst generischen, schnellen und vorzugsweise schlanken Alternative, die im fehlerfreien Betrieb geringen Overhead aufweist.

Diese Zusammenfassung der Dissertation „Zuverlässiger verteilter Speicher mit transaktionaler Konsistenz“ stellt ein leichtgewichtiges Verfahren zur Unterstützung von Persistenz in einem verteilten Betriebssystem mit transaktionaler Konsistenz vor und erläutert die erforderlichen Grundlagen im Bereich verteilter Systeme sowie die Synergien zwischen transaktionaler Konsistenz und Schnappschusserstellung. Die originale Schrift durchleuchtet auch den Lösungsraum für einen schnellen Wiederanlauf nach einem behebbaren Fehler und entwickelt die Verfahren, mit denen modifizierte Seiten im fehlerfreien Betrieb elegant eingesammelt und schnell gesichert werden können. Ebenso werden dort auch die besonders zu berücksichtigenden Zustände von Kern und Gerätetreibern diskutiert.

1 Einführung

1.1 Architektur verteilter Betriebssysteme

Verteilte Betriebssysteme (siehe [TS02]) sind zum Einsatz in einem Rechnerverbund konzipiert und enthalten im Vergleich zu nicht-verteilten Betriebssystemen (siehe [Sta03]) eine zum Teil deutlich ausgefeiltere Schnittstelle zur Kommunikation zwischen Rechnern. Oftmals können Objekte oder ganze Speicherseiten für den Anwendungsprogrammierer transparent zugegriffen und übertragen werden, wobei die Synchronisierung je nach System explizit durch den Programmierer erfolgen muss oder implizit durch das System vorgenommen wird. Persistenz (Dauerhaftigkeit) wird bei herkömmlichen Betriebssystemen über Dateien realisiert, die auf Medien wie einer Festplatte, einer CD oder einem USB-Stick abgelegt werden. Diese Medien können Daten auch ohne Stromversorgung über einen längeren Zeitraum, abhängig von den physikalischen Randbedingungen, halten.

Ziel verteilter Betriebssysteme ist neben der Vereinfachung der Kommunikation auch die Verwaltung von verteilbaren Betriebsmitteln. Meist jedoch können nicht alle Betriebsmittel für den Programmierer transparent verteilt werden, so dass hier eine explizite Kontrolle notwendig ist. Von großem Interesse ist üblicherweise die verteilte Verwaltung von CPU-Zeit und Hauptspeicher, um somit die verfügbaren und sich dynamisch ändernden Ressourcen optimal ausnutzen zu können.

Um den Anwendungsprogrammen auf allen Knoten eine einheitliche Basis anzubieten, ist ein Minimum an kompatiblen Schnittstellen erforderlich, und mit steigender Transparenz wird für diese Schnittstellen eine immer einheitlichere Installation benötigt. Eine vollständig einheitliche Sicht wird beim Single System Image realisiert (siehe [MLV+03]), hier wird der Anwendung auf jedem Rechner eine identische Sicht auf die vorhandenen Ressourcen, insbesondere Bibliotheken und Laufzeitstrukturen, ermöglicht. Im Falle einer vollständig identischen Installation aller beteiligten Rechner ist eine solche einheitliche Sicht zwar garantiert, jedoch ist es mitunter aufwendig oder unmöglich, alle Knoten mit gleichwertigen Bibliotheken auszurüsten. Eine mögliche Alternative zu dieser Art Pflege vieler Knoten besteht darin, statt vieler Systeme mit einheitlichen Bibliotheken auch das Betriebssystem zu verteilen. Somit kann der administrative Aufwand in Grenzen gehalten und der Anwendung dennoch ein Single System Image geboten werden (siehe [Goe05]).

1.2 Stand der Technik

Die derzeit eingesetzten Cluster stellen hohe Rechenleistungen für bestimmte Anwendungen zur Verfügung und können mit Hilfe spezieller Software und durch Hardware-Redundanz Ausfallsicherheit für Dienste in gewissem Umfang anbieten. Beispielsweise in Heartbeat (siehe [Rob99]) wird die Verfügbarkeit eines Dienstes dadurch sichergestellt, dass ausgefallene Maschinen durch sogenannte Backup-Maschinen ersetzt werden. Im „Tandem NonStop System“ (siehe [McE81]) hingegen wird für kleine Cluster Ausfallsicherheit garantiert, indem für jede nach außen angebotene Funktion mindestens zwei unabhängige Module bereitgestellt werden. Somit kann beim Ausfall eines Moduls das andere identische Modul sofort die Aufgaben des fehlerhaften Moduls übernehmen.

Parallel dazu hat sich ein allgemeinerer Ansatz zur Bereitstellung von Fehlertoleranz entwickelt: Durch die Erstellung von Schnappschüssen (Sicherungspunkte, Checkpoints) im fehlerfreien Betrieb kann ein System nach dem Auftreten eines Fehlers auf einen früher gesicherten Zustand zurückgesetzt werden. Dadurch gehen zwar die seit der Erstellung dieser Sicherung geänderten Daten verloren, jedoch ist der Hardware-Bedarf gegenüber vollständiger Redundanz erheblich reduziert (siehe [EAWJ96]). Die Bildung eines konsistenten Zustands auf der Ebene des Netzwerks ist aufwendig, da Nachrichten weder verloren gehen noch nach einer Rücksetzung doppelt auftauchen dürfen. Alternativ dazu gibt es (siehe [Kee89]) virtuelle verteilte Speicher (VVS, engl. Distributed Shared Memory, DSM), die den Zugriff auf ein Datum ortstransparent ermöglichen und somit die Kommunikation zwischen Anwendungen deutlich erleichtern. Die Konsistenzmodelle für VVS (siehe [Mos93]) sind für verschiedene Anwendungen optimiert und tendenziell eher restriktiv und einfach zu bedienen oder relaxiert und potenziell inkonsistent.

Der konsistente Zustand einer laufenden Anwendung besteht unabhängig vom Konsistenz-Modell nicht nur aus ihren Daten und offenen Netzwerk-Kanälen, sondern hängt auch von der transitiven Hülle ihres Datenpfades in anderen Anwendungen und im lokalen Betriebssystem ab. Die dadurch entstehende Menge an Daten ist immens und erhöht den Zeitaufwand und Speicherbedarf für einen Schnappschuss erheblich. Als Beispiel sei hier das Kerrighed System genannt, das die Erstellung eines Schnappschusses durch Erweiterung des Linux-Kerns (siehe [MLV+03]) realisiert. Durch den eingesetzten VVS ist der Datenzugriff feingranular und transparent möglich, so dass sich die Migration von Prozessen und die Erstellung von Schnappschüssen deutlich vereinfacht. Dennoch ist die Erstellung von Schnappschüssen durch die von Linux vorgegebene Architektur mit großem Aufwand verbunden, da die zur Anwendung gehörenden Daten im Kern bei der Erstellung eines Schnappschusses berücksichtigt werden müssen. Um die Kern-Zustände extrahieren und später wiederherstellen zu können, wird der so genannte Ghost-Mechanismus verwendet (siehe [VLM+05]), wofür tiefgreifende Änderungen des Linux-Kernels notwendig sind. Die Herausforderung liegt hier vor allem in der Wahl und Implementierung geeigneter Schnittstellen im Kern und außerdem in der Pflege des Systems, da die Modifikationen am Kern für jede zu unterstützende Kern-Version manuell anzupassen sind.

Neben der Entwicklung der Betriebssysteme besteht seit geraumer Zeit durch die Forschungstätigkeiten auf dem Gebiet der Datenbanken das Konzept der Transaktionen (siehe [HR83] und [Dad96]), das die Veränderung von Daten in mehrere Phasen einteilt und bestimmte Bedingungen an eine Transaktion knüpft. Die auf diesem Feld gewonnenen Erkenntnisse konnten im Rahmen des dieser Arbeit übergeordneten Plurix-Projekts der Universität Ulm auf verteilte Betriebssysteme für PC-Cluster übertragen werden.

1.3 Plurix

Das an der Universität Ulm entwickelte verteilte Betriebssystem Plurix verwendet einen seitenbasierten VVS, der als Halde (engl. Heap) organisiert ist. Die Verteilung wird durch Verteilung der für die Halde reservierten Seiten erreicht, so dass alle Knoten im Cluster eine identische Sicht auf die Halde haben. Der virtuelle gemeinsame Speicher auf Seitenbasis bietet zur Implementierung von orthogonaler Persistenz (siehe beispielsweise [Cla91]) eine gute Grundlage (siehe [Lie93]), da aus Sicht des Persistenzmoduls alle zu sichernden Einheiten gleich groß und durch ihre Adresse dauerhaft eindeutig identifizierbar sind. Des Weiteren wird dadurch Unabhängigkeit von den im Cluster eingesetzten Programmiersprachen sowie vom Aufbau der Objekte und der Halde erreicht, so dass der im Rahmen dieser Arbeit erstellte Prototyp auch die Anforderungen von Atkinson und Morrison an orthogonale Persistenz (siehe auch [AM95]) erfüllt. Statt wie in anderen DSM-Systemen einzelne Zugriffe auf Seiten zu protokollieren und zu publizieren, werden Transaktionen eingeführt, deren Ausführung entweder vollständig erfolgreich ist oder vollständig zurückgesetzt wird. Die verwendete transaktionale Konsistenz wird im nächsten Kapitel diskutiert.

2 Transaktionale Konsistenz

Wie in jedem verteilten System sind Fragen der Konsistenz und Synchronisierung von zentraler Bedeutung. Bekannte Verfahren für strenge Konsistenz (siehe [Wen03]) lassen sich in drei Kategorien unterteilen:

1. Pessimistischer Ansatz: Durch vorsorgliche Sperren wird jeglicher gleichzeitige Zugriff, der zu Inkonsistenz führen kann, ausgeschlossen.
2. Optimistische Synchronisierung: Alle Zugriffe werden vorerst gewährt und eventuell entstandene Konflikte werden durch Zurücksetzen von einzelnen Operationen oder von Gruppen von Operationen aufgelöst.
3. Ordnung mit Zeitstempel: Innerhalb eines Zeitintervalls auftretende Zugriffe werden gepuffert, unter Beachtung von Zeitstempeln sowie Abhängigkeiten geordnet und erst dann tatsächlich ausgeführt.

Bei Verwendung optimistischer Synchronisierung werden Kopien sowie konkurrierende Zugriffe nicht von vornherein ausgeschlossen, sondern bei schreibenden Zugriffen wird nachträglich geprüft, ob Konflikte vorliegen. Diese werden durch Zurücksetzen einzelner Operationen aufgelöst, was die Rücksetzbarkeit von Operationen beziehungsweise die Konfliktfreiheit von nicht rücksetzbaren Operationen impliziert. Für den Hauptspeicher kann Ersteres sehr einfach durch Schattenkopien oder Logs realisiert werden, beim Zugriff auf Geräte ohne Rücksetzungsfunktion wie zum Beispiel Druckern ist jedoch durch geeignete Maßnahmen auf Konfliktfreiheit zu achten (siehe [OV91]). Dieses Verhalten findet sich in ähnlicher Weise auch bei Transaktionen, wobei deren ACID-Definition (siehe [Dad96]) über die Festlegung der Konsistenz hinausgeht:

1. Atomarität (Atomicity): Die auch als „all-or-nothing“ und „failure-atomicity“ bezeichnete Eigenschaft beschreibt den Umstand, dass Transaktionen entweder vollständig ausgeführt werden (Erfolg, „commit“) oder keinerlei Änderungen des Gesamtsystems hinterlassen (Abbruch, „abort“).
2. Konsistenz (Consistency): Im klassischen Fall überführt eine Transaktion das Gesamtsystem von einem konsistenten Zustand in einen neuen konsistenten Zustand. Systeme mit zwischenzeitlichem Commit (siehe [Bla90]) sind möglich, jedoch muss die Konsistenz in diesen Systemen vom Programmierer meist selbst gewährleistet werden.
3. Isolierung (Isolation): Während der Laufzeit einer Transaktion ändern sich ihre Eingabedaten nicht und ihre Zwischenergebnisse sind für andere Transaktionen nicht sichtbar. Insbesondere resultiert hieraus die Serialisierbarkeit (auch „concurrency-atomicity“ genannt) aller Transaktionen in einem System.
4. Dauerhaftigkeit (Durability): Die Ergebnisse einer erfolgreichen Transaktion müssen dauerhaft sein, also auch Fehler überleben (siehe auch [Sta04]). Dies erfordert für die Praxis einerseits die Betrachtung möglicher Fehler und andererseits die Spezifikation von zu tolerierenden Fehlern.

In Datenbanksystemen wie Oracle wird die Sicherung auf einem nicht-flüchtigen Speicher als notwendig und ausreichend angesehen. Verschärft wird dies zum Beispiel bei Buchungssystemen in Geldinstituten mit redundanter Sicherung, abgeschwächt für höheren Durchsatz (Entkopplung von Transaktion und Sicherung) mit verzögerten Schreibzugriffen wie bei ISAM-Tabellen unter MySQL (siehe [WT03]).

2.1 Gerätetreiber

Wie auch bei nicht-verteilten transaktionalen Systemen erfordert die Integration von Gerätetreibern besonderes Augenmerk, da sich die Geräte üblicherweise außerhalb des transaktionalen Raums befinden. Das System muss dafür Sorge tragen, dass sich die Geräte beim Abbruch einer Transaktion wieder in dem konsistenten Zustand befinden, der zum logischen Zeitpunkt vor der abgebrochenen Transaktion passt. Die in [Fre06] entwickelte Systematik erlaubt eine effiziente Verwaltung solcher Informationen, so dass bei einer Rücksetzung nach einem Fehler auch die Gerätetreiber in den zum transaktionalen Speicher konsistenten Zustand versetzt werden können.

2.2 Integration von Dauerhaftigkeit

Bei der Integration von Dauerhaftigkeit in ein bestehendes System sind zum einen die Anforderungen an die Dauerhaftigkeit, zum anderen aber auch die Schnittstellen zum bestehenden System zu erörtern. Für eine Implementierung von Dauerhaftigkeit in einem transaktionalen verteilten Speicher ergeben sich folgende Ansatzpunkte:

1. Die Speicherverwaltung: Beim Abschluss einer Transaktion sorgt die lokale Speicherverwaltung bei der Übernahme der durch die Transaktion veränderten Werte für eine dauerhafte Sicherung auf dem eigenen Knoten. Die Konsistenzierung des Hauptspeichers wird also mit einer lokalen Sicherung verbunden.

2. Das Protokoll: Beim erfolgreichen Eintritt in die Commitphase werden alle publizierten Modifikationen lokal gesichert oder andere Teilnehmer innerhalb oder außerhalb des Clusters zur Replizierung aufgefordert.

3. Das Netzwerk: Bei oder nach dem Versand von publizierten Modifikationen werden die modifizierten Daten angefordert und außerhalb des transaktionalen Systems gesichert.

Lokale Sicherungen wie bei Möglichkeiten (1) und (2) sind schnell und ohne Belastung des Gesamtsystems realisierbar, bergen jedoch beim Ausfall eines beliebigen Knotens im Cluster die Gefahr des vollständigen Datenverlustes, da nur alle lokalen Sicherungen gemeinsam eine konsistente Version der transaktionalen Daten bilden können. Bei einem auftretenden Fehler müssen die über mehrere Knoten verteilten Zustände zeit- und kommunikationsintensiv analysiert werden, um die relevante Datenbasis bestimmen zu können.

Unter Verwendung der Möglichkeit (3) ergeben sich mehrere Konsequenzen, die je nach Anwendungsgebiet unterschiedlich bewertet werden müssen und nur im Kontext als Vor- oder Nachteil bezeichnet werden können. Wesentlicher Bestandteil einer Integration von Dauerhaftigkeit auf Basis der Kommunikation über das Netzwerk besteht in der Möglichkeit einer Entkopplung des Clusterbetriebs von dediziert zur Verfügung gestellten Sicherungsrechnern.

Die Entkopplung des Transaktionsabschlusses von der Sicherstellung der Dauerhaftigkeit ist je nach Anwendung erfreulich, da sich für diesen Fall eine deutlich effizientere Sammlung der Daten realisieren lässt, oder nachteilig, da im Fehlerfall auch bestätigte Modifikationen von Transaktionen verloren gehen können, was der Semantik von Datenbank-Transaktionen hinsichtlich der Dauerhaftigkeit widerspricht. Prinzipiell lässt sich der Abschluss einer Transaktion auch an die Sicherung im Pageserver koppeln, wodurch die von Datenbanken geforderte Bedingung der Integration erfüllt werden könnte. Jedoch erfordert dies den permanenten Einsatz des Pageservers und eine Verankerung der Kommunikation mit dem Pageserver im Protokoll, zudem wird der Abschluss jeder Transaktion entscheidend verlangsamt.

Beim Zurücksetzen des Clusters ist ein außerhalb des Clusters angesiedelter Pageserver imstande, dem neu startenden Cluster ohne dessen Hilfe ein konsistentes Abbild zur Verfügung zu stellen. Wären die Daten eines vollständigen Abbildes auf alle Einzelknoten verteilt, so wäre eine globale Sicht auf die verteilt gesicherten Daten erforderlich, was einen hohen Rechen- sowie Kommunikations- und somit Zeitaufwand beim Sammeln und Auswerten der Daten bedeuten würde. Zudem wäre diese globale Sicht bei einem mittel- oder langfristigen Ausfall eines Knotens nur mit Hilfe von zusätzlicher Replikation innerhalb des Clusters möglich. Die dafür benötigte Kommunikation ist mindestens so hoch wie die Kommunikation mit einem Pageserver, belastet jedoch die übrigen Maschinen im Cluster. Das Abbild sollte also außerhalb der abzusichernden Rechner liegen, um bei dauerhaften Hardwarefehlern sofort auf eine gültige Datenbasis zurückgreifen zu können (siehe [LDN97]).

Weiterhin kommuniziert ein eigenständiger Pageserver mit dem Cluster ausschließlich über das Netzwerk-Protokoll, so dass dies die einzige Schnittstelle zwischen Pageserver und Cluster darstellt. Diese Schnittstelle ist klar definiert und überschaubar, wodurch eine einfache und effiziente Struktur für den Pageserver ermöglicht wird.

3 Synergien zwischen transaktionaler Konsistenz und Checkpointing

Die Anforderungen für Schnappschüsse lassen sich mit Hilfe von transaktionaler Konsistenz effizient und elegant umsetzen. Dieses Kapitel beleuchtet mehrere besonders auffällige Aspekte der transaktionalen Konsistenz:

1. Die Änderungsfrequenz der nach außen sichtbaren Daten ist niedrig.
2. Alle nach außen sichtbaren Daten sind Bestandteil des letzten konsistenten Zustands.

3. Trotz äußerst geringem Overhead im Betrieb zur Übertragung der für eine Sicherung erforderlichen Seiten ist eine sehr schnelle Rücksetzung möglich.

4. Die Funktionen zur Rücksetzung von Daten sind Bestandteil des zugrunde liegenden Systems.

In Systemen wie Ivy, Treadmarks oder Kerrighed (siehe auch [Li88], [KCDZ94] und [FLS+05]) werden schreibende Zugriffe auf Seiten ermöglicht, indem der schreibenden Maschine für eine kurze Zeit exklusiver Zugriff auf diese Seite gestattet wird und die ansonsten im Cluster vorhandenen Daten sofort (strikte Konsistenz), mit kurzer Verzögerung (sequenzielle Konsistenz) oder explizit zu einem späteren Zeitpunkt (abgeschwächte Konsistenz) invalidiert werden. Solange die Zeitspanne für den schreibenden und somit exklusiven Zugriff nicht abgelaufen ist, können die Anfragen auf diese Seite nicht beantwortet werden. Danach ist für einen schreibenden Zugriff eine erneute Absicherung der Exklusivität erforderlich. Dagegen werden bei transaktionaler Konsistenz Anfragen auf Seiten immer mit der zuletzt veröffentlichten Version beantwortet und beliebig viele Schreibzugriffe, auch auf mehrere Seiten, am Ende einer Transaktion gebündelt publiziert. Somit ergibt sich eine niedrige Änderungsfrequenz (1) der nach außen sichtbaren Daten. Dies hat insbesondere Auswirkungen auf Seiten, die von einer Station geschrieben und von vielen Stationen gelesen werden, da diese über einen vergleichsweise langen Zeitraum auf die zuletzt publizierte, also aktuelle Version einer Seite zugreifen können und diese erst nach Invalidierung durch eine abschließende Transaktion erneut anfordern müssen.

Dieses Verhalten beeinflusst insbesondere das Einsammeln von Daten durch den Pageserver (2). Denn während der Ausführung von Transaktionen im Cluster sind alle bisher publizierten Daten vom Pageserver aus erreichbar. Somit kann dieser die aktuell gültige Version des Gesamtsystems nebenläufig zu den Änderungen vieler Transaktionen ermitteln und sichern, da erst bei deren Abschluss neue Versionen aus den Ergebnissen der Transaktionen gebildet werden. Im Gegensatz dazu wird in anderen Systemen häufig versucht, durch Verfahren zur Koordinierung (siehe [EAWJ96], [Lam78], [CL85] oder [Agb02]) einen punktuell gültigen Schnappschuss zu erstellen, wobei außer dem Systemzustand auch die noch auf dem Kommunikationsmedium befindlichen Nachrichten berücksichtigt werden müssen (siehe [TS02]). Bei unabhängigen Schnappschüssen kann es vorkommen, dass mit einer Kombination der Schnappschüsse kein global konsistentes Abbild gefunden werden kann, so dass bei einem Fehler auf den initialen Zustand zurückgesetzt werden muss (Domino-Effekt, siehe [Ran75]). Dies ist bei transaktionaler Konsistenz vermeidbar, da hier auf einfache Weise ein vollständig konsistentes Abbild erstellt werden kann. Zwei wesentliche Unterschiede bestehen also zum Ersten in der Lebensdauer der nach außen sichtbaren Versionen, die sich im Falle von transaktionaler Konsistenz auf den Zeitraum zwischen zwei erfolgreichen Abschlüssen von Transaktionen erstreckt und somit konsistentes Checkpointing (siehe [EJZ92]) vereinfacht, während die Aktualität in nicht transaktionalen Systemen beim nächsten Ändern eines Datums oder dem Austausch einer Nachricht verloren geht. Zum Zweiten erfordert die Bestimmung einer konsistenten Datenbasis bei nicht transaktionalen Systemen spezielle Synchronisierung oder Protokollierung, während bei transaktionaler Konsistenz garantiert wird, dass alle sichtbaren Daten zu einem konsistenten Abbild gehören.

Üblicherweise muss zwischen Geschwindigkeit im fehlerfreien Betrieb und Aktualität der Daten beim Checkpointing ein Kompromiss gefunden werden (siehe [DN04]). Trotz niedriger Belastung des Clusters im laufenden Betrieb ermöglicht der implementierte Prototyp dennoch eine schnelle Rücksetzung im Fehlerfall (3), wie durch die Messungen in [Fre06] belegt und im nächsten Kapiel zusammengefasst wird.

Der Aufwand zur Bereitstellung dieser Funktionalität ist bei Systemen, die Transaktionen unterstützen, äußerst gering (4), da bereits aufgrund der Rücksetzbarkeit von Transaktionen alle notwendigen Algorithmen zur Rücksetzung von Anwendungen vorhanden sind. Mit der integrierten logischen Zeit (siehe auch [Wen03]) wird automatisch eine gültige Datenbasis bei der Konsistenzierung verwendet, was in anderen Systemen spezielle Synchronisierung der Maschinen oder Analyse der verschickten Nachrichten erfordert (siehe [CL85] und [EAWJ96]).

4 Fazit

Die in herkömmlichen Betriebssystemen vorhandene Trennung zwischen Kern und Anwendungen kann alternativ unter Verwendung einer typsicheren Sprache aufgehoben werden, wie durch die Forschungen des Oberon-Projekts belegt wurde. Auch im verteilten Fall lassen sich die Eigenschaften Atomarität, Konsistenz und Isolierung in einem fehlerfrei arbeitenden System effizient implementieren. Wenn Fehler auftreten, ergeben sich allerdings Herausforderungen bei der Wiederherstellung eines konsistenten Zustands, deren Lösungsansätze bisher jedoch in einem sehr hohen Aufwand auch im fehlerfreien Betrieb resultierten oder sehr aufwendige Analysen im Falle eines Fehlers erforderten.

Die sich daraus ergebenden Fragestellungen werden in der zu dieser Zusammenfassung gehörenden Arbeit [Fre06] ausführlich diskutiert und anhand des ausgeleuchteten Lösungsraums zu einem leichtgewichtigen und dennoch hocheffizienten Verfahren entwickelt, um Schnappschüsse in einem verteilten und auf Transaktionen basierenden Betriebssystem unter Berücksichtigung der Gegebenheiten heutiger Hardware zu erstellen und somit Persistenz für solche Systeme anzubieten.

Anhand des implementierten Prototyps wird belegt, dass die bisher bestehende Aufwandsgrenze zur Erstellung eines Schnappschusses selbst dann überwunden werden kann, wenn die Daten des Betriebssystems und ausgewählte Informationen der Gerätetreiber ebenfalls Bestandteil des Schnappschusses sind. Somit kann auch beim vollständigen Ausfall von Knoten die gesamte Umgebung der laufenden Transaktionen einschließlich der Zustände der Geräte wiederhergestellt werden. Der implementierte Prototyp erlaubt eine Rücksetzung des Systems mit Fast Ethernet in etwa 250 Millisekunden, wobei mit einem zusätzlichen Aufwand von nur etwa fünf Millisekunden pro Knoten eine zur Anzahl der Knoten annähernd lineare Skalierung erreicht wurde. Die Messungen belegen, dass die Erstellung auch in äußerst kurzen Abständen von unter vier Sekunden möglich ist und dabei den Cluster dennoch nicht über Gebühr belastet: im Falle der gemessenen Anwendungen, einem Raytracer, verbleiben über 96% der verfügbaren Zeit für die Anwendung selbst.

Literatur

[Agb02] A. Agbaria. *Reliability in High Performance Distributed Computing Systems*. Dissertation am Israel Institute of Technology, Haifa, Israel, 2002.

[AM95] M.P. Atkinson und R. Morrison. *Orthogonally Persistent Object Systems*. International Journal on Very Large Data Bases 4, p319-401, 1995.

[Bla90] A.P. Black. *Understanding Transactions in the Operating System Context*. Proceedings of the 4th Workshop on ACM SIGOPS European Workshop, p1-4, 1990.

[CL85] K.M. Chandy und L. Lamport. *Distributed Snapshots: Determining Global States of Distributed Systems*. ACM Transactions on Computer Systems, vol3(1), p63-75, 1985.

[Cla91] S.M. Clamen. *Data Persistence in Programming Languages, A Survey*. CMU-CS-91-155, Pittsburgh, 1991.

[Dad96] P. Dadam. *Verteilte Datenbanken und Client/Server-Systeme - Grundlagen, Konzepte, Realsierungsformen*. Springer-Verlag, Heidelberg, 1996.

[DN04] T. Dumitras und P. Narasimhan. *An Architecture for Versatile Dependability*. DSN Workshop on Architecting Dependable Systems, Italien, 2004.

[EAWJ96] E.N. Elnozahy, L. Alvisi, Y.M. Wang und D.B. Johnson. *A Survey of Rollback-Recovery Protocols in Message-Passing Systems*. TR, Carnegie Mellon University, 1996.

[EJZ92] E.N. Elnozahy, D.B. Johnson und W. Zwaenepoel. *The Performance of Consistent Checkpointing*. Proceedings of Reliable Distributed Systems, p39-47, 1992.

[FLS+05] S. Frenz, R. Lottiaux, M. Schöttner, C. Morin, R. Göckelmann und P. Schulthess. *A Practical Comparison of Cluster Operating Systems Implementing Sequential and Transactional Consistency*. Proceedings of the 6th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP), Australien, 2005.

[Fre06] S. Frenz. *Zuverlässiger verteilter Speicher mit transaktionaler Konsistenz*. Dissertation an der Universität Ulm, 2006.

[Goe05] R. Goeckelmann. *Speicherverwaltung und Bootstrategien für ein Betriebssystem mit transaktionalem verteilten Heap*. Dissertation an der Universität Ulm, 2005.

[HR83] T. Haerder und A. Reuter. *Principles of Transaction-Oriented Database Recovery*. Computing Surveys, vol15(4), p287-317, 1983.

[KCDZ94] P. Keleher, A.L. Cox, S. Dwarkadas und W. Zwaenepoel. *TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems*. Proceedings of the Winter 1994 USENIX Conference, 1994.

[Kee89] J.L. Keedy. *The MONADS-PC System: A Programmers Overview*. Report 8/89, Universität Bremen, 1989.

[Lam78] L. Lamport. *Time, Clocks, and the Ordering of Events in a Distributed System*. Communications of the ACM, vol21(7), p558-565, 1978.

[LDN97] J.L. Lin, M.H. Dunham und M.A. Nascimento. *A Survey of Distributed Database Checkpointing*. Distributed and Parallel Databases, vol5(3), p289-319, 1997.

[Li88] K. Li. *IVY: A Shared Virtual Memory System for Parallel Computing*. Proceedings of the International Conference on Parallel Processing, 1988.

[Lie93] J. Liedtke. *A Persistent System in Real Use*. Proceedings of the International Workshop on Object-Orientation in Operating Systems (IWOOOS), 1993.

[McE81] D. McEvoy. *The Architecture of Tandems NonStop System*. Proceedings of the ACM81 Conference, p245, 1981.

[MLV+03] C. Morin, R. Lottiaux, G. Vallee, P. Gallard, G. Utard, R. Badrinath und L. Rilling. *Kerrighed: a Single System Image Cluster Operating System for High Performance Computing*. Proceedings of Europar 2003: Parallel Processing, vol2790 of Lecture Notes in Computer Science, p1291-1294, 2003.

[Mos93] D. Mosberger. *Memory Consistency Models*. TR, University of Arizona, 1993.

[OV91] M.T. Öezsu und P. Valduriez. *Principles of Distributed Database Systems*. Prentice-Hall International, 1991.

[Ran75] B. Randell. *System Structure for Software Fault-Tolerance*. IEEE Transactions on Software Engineering, vol1(2), 1975.

[Rob99] A. Robertson. *Linux High Availability*. Linux High Availability Project, 1999.

[Sta03] W. Stallings. *Betriebssysteme: Prinzipien und Umsetzung*. Übersetzung der 4. Auflage, Prentice Hall, 2003.

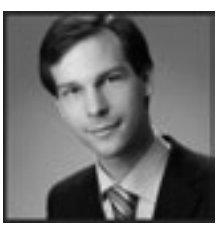
[Sta04] J. Stanton. *Distributed Operating Systems*. Lecture CS 251, 2004/6, Department of Computer Science, George Washington University, 2004.

[TS02] A.S. Tanenbaum und M.v. Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2002.

[VLM+05] G. Vallee, R. Lottiaux, D. Margery, C. Morin und J.-Y. Berthou. *Ghost Process: a Sound Basis to Implement Process Duplication, Migration and Checkpoint/Restart in Linux Clusters*. 4th International Symposium on Parallel and Distributed Computing, Lille, Frankreich, p97-104, 2005.

[Wen03] M. Wende. *Kommunikationsmodell eines verteilten virtuellen Speichers*. Dissertation an der Universität Ulm, 2003.

[WT03] L. Welling und L. Thomson. *MySQL Tutorial: A concise introduction to the fundamentals of working with MySQL*. MySQL Press, 2003.



Stefan Frenz wurde am 16. April 1978 in Ulm/Donau geboren. Gegen Ende der Stuttgarter Gymnasialzeit besuchte er die Informatik-AG für mathematisch begabte Schüler am Fraunhofer Institut in Vaihingen. Noch vor dem Informatik-Studium mit Nebenfach Mathematik an der Universität Ulm ab 1997 arbeitete er am DaimlerBenz-Forschungszentrum, während des Studiums dann unter anderem am Lehrstuhl für Stochastik im Projekt zur Simulation und graphischen Darstellung mehrdimensionaler stochastischer Modelle. Nach dem sehr guten Abschluss des Studiums 2002 nahm er das Angebot einer Anstellung am Lehrstuhl für Verteilte Systeme an, wo seine Aufgaben auch die Qualitätssicherung der im Team erstellten Betriebssystem-Module umfaßten. Er entwickelte einen Pageserver zur Gewährleistung von Ausfallsicherheit in einem verteilten Betriebssystem mit transaktionaler Konsistenz und schloss seine Dissertation im Mai 2006 mit Auszeichnung ab. Seit Oktober 2006 arbeitet er bei der Robert Bosch GmbH in der Forschung und Vorausentwicklung in Schwieberdingen.