Proseminar Speicher- und Dateisysteme

Datenintegrität

von
Christian Rosenberg

Inhaltsverzeichnis

1.1. Definition Datenintegrität

Bezüglich zur Definition von Datenintegrität kann man zunächst festhalten, dass keine einheitliche Formulierung derselbigen existiert. Es gibt jedoch diverse Ansätze, welche im Folgenden genannt werden. Die ITSEC, welche als Standard für Bewertungen von Sicherheit in der Informationstechnologie gilt, beschreibt Datenintegrität als „Verhinderung unautorisierter Modifikation von Informationen“. Das Bundesamt für Sicherheit in der Informationstechnik definiert Datenintegrität als „Korrektheit von Daten und Systemen“. Ein weiterer Ansatz stammt vom Experten für Informationssysteme Professor Joachim Biskup, welcher Datenintegrität in vier Arten von Integrität zerlegt. Demnach müssen alle vier Voraussetzungen gegeben sein, damit Datenintegrität gilt. Diese Voraussetzungen lauten:

1. Korrekter Inhalt: Sachverhalte aus der realen Welt sollen korrekt abgebildet sein. Bewerkstelligt wird dies konkret durch Integritätsbedingungen, welche voraussetzen, dass Daten sich jederzeit in einem konsistenten Zustand befinden
2. Unmodifizierter Zustand: Nach einem Datenaustausch sind die Daten im gleichen Zustand wie zuvor. Anders ausgedrückt bedeutet das, dass Daten bzw. Programme weder absichtlich von Dritten manipuliert noch durch andere unbeabsichtigte äußere Einflüsse wie zum Beispiel Störungen verändert werden darf.
3. Erkennen von Modifikation: Es können Veränderungen an Daten auftreten, die sich nicht vermeiden lassen. Diese sollen jedoch zumindest vom Anwender bzw. System erkannt werden.
4. Temporale Korrektheit: Es müssen Bedingungen eingehalten werden, die zeitliche Abläufe beim Datenaustausch betreffen. Gemeint sind hiermit beispielsweise die Einhaltung der Reihenfolge von Datenpaketen beim Austausch oder das Festlegen einer maximalen Verzögerungszeit.

1.2. Methoden zur Bewerkstelligung von Datenintegrität

1.2.1 Sequenznummern

Sequenznummern werden in der Datenübertragung genutzt und dienen der Vermeidung von Duplikaten sowie der Einhaltung der Reihenfolge von Datenpaketen. Sie werden bei Netzwerkprotokollen wie TCP verwendet und funktionieren nach einem simplen Schema: Eine zu übertragende Datei wird in Pakete aufgeteilt, welche alle mit einer Nummer (nämlich der Sequenznummer) versehen werden. Nach Erhalt eines Pakets kann somit immer überprüft werden, ob ein Paket bereits mehrfach empfangen oder eventuell Pakete ausgelassen wurden.

1.2.2. Quittierungsmeldugen

Quittierungsmeldungen finden ebenfalls bei Datenübertragungen Verwendung. Wie der Name andeutet, bestätigen Quttierungsmeldungen den Erhalt oder die vollständige Verarbeitung einer Datei. Diese werden in Form von Signalen wie beispielsweise Datenblöcke bei TCP-Protokollen

oder per Steuerzeichen übermittelt. Steuerzeichen sind Zeichen, welche nicht durch Buchstaben oder Ziffern dargestellt werden. Quittierungsmeldungen werden zum Beispiel durch die Tastenkombination Strg + F gebildet. Bei erfolgreicher Übertragung bzw. Verarbeitung wird ein sogenanntes ACK-Signal (Abkürzung für „Acknowledgement“, engl. für „Bestätigung“) übermittelt. Entsprechend wird bei Auftreten von Fehlern ein NAK-Signal übermittelt und eine Wiederholung der Übertragung angefordert.

1.2.3. Prüfsummen (einfache Verfahren)

Prüfsummen spielen in der Datensicherung sowie -übertragung eine Rolle und lassen sich in einfache sowie komplexe Verfahren unterteilen. Bei der Bildung von Prüfsummen durch einfache Verfahren betrachtet man Komponenten von Daten, typischerweise die Bitfolge, und wandelt diese zunächst in Zahlen um. Anschließend werden diese mit einem konstanten Faktor multipliziert und nacheinander aufaddiert. Der hieraus entstandene Wert bildet die Prüfsumme. Im Anschließenden wird die Prüfsumme gemeinsam mit der zu übertragenden Datei versendet. Der Empfänger bildet aus der empfangenen Datei die Prüfsumme und vergleicht diesen mit der vom Versender erzeugten. Bei Übereinstimmung ist die empfangene Datei mit hoher Wahrscheinlichkeit korrekt. Einfache Verfahren stellen lediglich einen Schutz vor unbeabsichtigter Veränderung von Daten dar und werden z.B. für ISBN-Nummern oder für Prüfsummen bei Kreditkarten genutzt. Komplexere Verfahren wie der CRC und Hash-Verfahren werden im Folgenden erklärt.

1.2.4. Cyclic Redundancy Check (CRC)

Die Abkürzung CRC bedeutet übersetzt „Zyklische Redundanzprüfung“. CRC-Werte werden mithilfe von Polynomdivision erzeugt. Das Verfahren soll vor Zufallsfehlern schützen und läuft nach folgendem Schema ab:

1. Eine Bitfolge wird als binäres Polynom betrachtet
2. Ein Generatorpolynom wird beliebig gewählt. Der Datenfolge werden eine bestimmte Anzahl an Nullen angehängt, welche sich aus dem Grad des Generatorpolynoms minus 1 ergibt. Anschließend dividiert man das Polynom der Bitfolge durch das Generatorpolynom
3. Der hieraus entstandene Rest der Rechnung wird der zu übertragenden Datenfolge angefügt
4. Die empfangene Datenfolge wird schließlich erneut durch das Generatorpolynom dividiert. Logischerweise muss sich als Rest dieser Rechnung Null ergeben, um eine korrekte Übertragung zu bestätigen.

Um dieses sehr abstrakte Verfahren zu veranschaulichen, sei folgendes Beispiel genannt (Quelle: wikipedia.de):

Beispiel CRC erzeugen :
1101100000 (Datenfolge + n Nullen)
110101 (Generatorpolynom)

0000110000
110101
---
101 (Rest) → 00101 (n-Stellig)

Erläuterung:
Der Grad des Generatorpolynoms ist in diesem Beispiel 6. Es werden demnach 5 Nullen (6-1 = 5) an die zu übertragende Datenfolge angehängt. Die Polynomdivision ist beim CRC-Verfahren sehr simpel. Von links beginnend werden die jeweiligen Ziffern des Divisors (entspricht dem Generatorpolynom) mit denen des Dividenden (also der Bitfolge) verglichen. Bei Übereinstimmung lautet der Rest 0, ansonsten 1. Die Division wird solange durchgeführt, bis der Grad des Rests kleiner ist als der des Generatorpolynoms. Der Rest (in diesem Fall 101) muss in diesem Fall wieder 5-stellig sein, da das Generatorpolynom 6-stellig ist. Deswegen werden dem Rest vorne zwei Nullen angefügt. Es ergibt sich der neue Rest: 00101. Dieser Rest wird nun der ursprünglichen Datenfolge (ohne Nullen) angefügt und übertragen. Die folgende Division wird vom Empfänger durchgeführt:

Beispiel CRC-Prufung:
1101100101 (Bitfolge + CRC)
110101 (Generatorpolynom)

110101
110101

000000

In diesem Fall lautet der Rest Null. Die Übertragung war somit aller Wahrscheinlichkeit nach erfolgreich. Jedoch lässt sich das nicht hundertprozentig sagen. Es können nämlich folgende Fälle beim CRC-Verfahren eintreten:

1. Der Rest ist gleich Null und die Nachricht wurde korrekt übertragen.

2. Der Rest ist gleich Null und die Nachricht ist fehlerhaft. In diesem Fall ist das Fehlerpolynom entweder ein Vielfaches vom Generatorpolynom oder der Fehler liegt sowohl im CRC als auch im Datenteil.

3. Der Rest ist ungleich Null und die Datei ist fehlerhaft.

4. Der Rest ist ungleich Null und die übertragene Datei ist richtig. Dieser Fall tritt ein, wenn nur der CRC-Wert fehlerhaft übertragen wurde. Dies ist jedoch sehr unwahrscheinlich, da der CRC im Gegensatz zur eigentlichen Datei immer verhältnismäßig kurz ausfällt.

Abschließend ist noch festzuhalten, dass für das CRC-Verfahren folgende Voraussetzungen gegeben sein müssen: Empfänger und Sender einer Datei müssen das gleiche Generatorpolynom sowie Rechenverfahren benutzen und dem Empfänger muss bekannt sein, an welcher Stelle der Datenfolge sich der CRC-Wert befindet.

1.2.5. Hash-Verfahren

Das Verb „to hash“ stammt aus dem Englischen und bedeutet „zerhacken“. Entsprechend versteht man unter einer Hashfunktion das Prinzip, eine sehr große Quellmenge in eine deutlich komprimierte Zielmenge umzuwandeln. Hashfunktionen sind Algorithmen statt mathematischen

Funktionen. Damit wird sichergestellt, dass eine Hashfunktion nach endlich vielen Schritten terminiert und nicht möglicherweise eine Endlossschleife entsteht. Das Ziel einer Hashfunktion ist, dass bei unterschiedlichen Eingabewerten stets unterschiedliche Ausgabewerte entstehen. Dies ist das wichtigste Kriterium für die Güte einer Hashfunktion. Einen identischen Ausgabewert als Ergebnis zwei unterschiedlicher Eingabewerte bezeichnet man als Kollision, welche zu vermeiden ist. Eine weit verbreitet Hashfunktion, die Tiger-Funktion, liefert beispielsweise folgende Hashwerte bei diesen Eingaben (Quelle: wikipedia.de):

Eingabe: „Franz jagt im komplett verwahrlosten Taxi quer durch Bayern“
→ Hashwert: 4df42db66c8d84269d4b7157b92a87be717aa1a5834a3050

Eingabe: „Frank jagt im komplett verwahrlosten Taxi quer durch Bayern“
→ Hashwert: 9cee0eb7b596ba0f435d42c33ddf8eff7fabb86922aa4bc6

Es ist zu erkennen, dass bei dieser Funktion eine sehr kleine Änderung des Eingabewerts einen gänzlich unterschiedlichen Hashwert liefert.

In der Informationssicherheit werden Hashfunktionen für sogenannten Hashbäumen verwendet. Ein Hashbaum ist eine Datenstruktur, die einen Baum aus Hashwerten von Datenblöcken bildet. Ursprünglich hauptsächlich für Signaturen verwendet, stellen Hashbäume mittlerweile zusätzlich einen Schutz vor Datenmanipulation dar. Aus den Datenblöcken werden Hashwerte gebildet, welche sich am unteren Ende des Hashbaums befinden. Übergeordnete Glieder des Baums sind Hashwerte aller direkt untergeordneten Hashwerte. Konkret bedeutet das, dass jemand, der im Besitz des obersten Hashwerts (dem Top-Hash) ist, mithilfe diesen jeder Datenblock auf Unverfälschtheit überprüft werden kann. Ein großer Vorteil von Hashbäumen im Vergleich zu anderen Verfahren wie zum Beispiel Hash-Listen ist ihre Effizienz.

2.1. Memory Corruption

Memory Corruption ist definiert als eine unbeabsichtigte Änderung des Speicherinhalts durch Programmierfehler. Beim Auftreten von Memory Corruption befindet sich im Hauptspeicher Inhalt, deren Wert um einen oder mehrer Bits verändert ist. Ist Memory Corruption durch Softwarefehler bedingt, so ist kein Unterschied zu einer Disk Corruption bei Lesen eines Datenträgerblocks feststellbar. Memory Corruption ist eine nicht zu unterschätzende Gefahr für den Nutzer, da die Speichersicherheit durch diese beeinträchtigt wird. Folgen einer Memory Corruption sind ein Programmabsturz oder unerwartetes Verhalten des Programms. Das zurückverfolgen eines Fehlers ist insbesondere problematisch, da zum Einen die Quelle bzw. Ursache des Fehlers und deren Ausprägung weit voneinander entfernt sein können. Zum Anderen können die Ausprägungen sehr verschieden sein, wodurch die Ursache schwer zu ermitteln ist.

Folgende Problemfelder sind die häufigsten Ursachen für Memory Corruption:

Physikalische Gegebenheiten, wie zum Beispiel Alpha-Teilchen, die auf Hardware treffen und den Speicher verändern oder die Beeinflussung elektrischer Ströme durch kosmische Strahlen können ständig auftreten und sind unvermeidbar.

Weiterhin können softwaretechnische Probleme auftreten. *Bugs* in Software, die auf Speicherinhalte zugreift und diese ändert, treten häufig bei instabilen Programmiersprachen wie C++ auf.
Ein *dangling Pointer oder hängender Zeiger* ist ein ungültiger Wert, der einem nicht vorhandenen Speicherbereich zugeordnet ist und verursacht Programmabstürze oder im schlimmsten Fall sogar Systemschäden.
Bei einem *Buffer-Overflow* ist eine große Datenmenge in einem zu kleinen reservierten Speicherbereich (Puffer) gesichert. Die Palette an negativen Auswirkungen ist sehr groß; Programmabstürze, Verfälschung von Daten, oder Beschädigung von Datenstrukturen in der Laufzeitumgebung sind einige davon.

Memory Corruptions werden nach Härtegrad in zwei Klassen aufgeteilt, in *Soft Errors* und *Hard Errors*. Soft Errors sind zufällige Änderungen der Bitwerte und nur sehr kurzlebig. Sie sind deshalb weniger gefährlich. Hard Errors sind langfristige Fehler mit schwerwiegenden Konsequenzen.

Um Memory Corruption entgegenzuwirken und vorzubeugen, wurden diverse Verfahren entwickelt. *Error Correction Codes (ECC)* sind Algorithmen zum Finden beschädigter Dateien und zur Fehlerbehebung. Gewöhnliche Algorithmen wie SEC/DED (Single Error Correct/Double Error Detect) sind zumeist in Speichereinheit bereits integriert. Mit diesen Algorithmen können jedoch lediglich 94% der Fehler können behoben werden. Nach einer Studie des Unternehmens Hewlett & Packard hat ein Gigabyte Arbeitsspeicher 3400 FIT. Ein FIT entspricht einem Fehler in 10^9 Stunden Betriebszeit. Anschaulich bedeutet das, dass 900 Fehler in 10000 Rechnern entstehen würden, nachdem diese 3 Jahre in Betrieb waren. Neben den ECCs können fortschrittlichere Algorithmen wie *Chipkill* sogar Multi-Bit-Fehlern standhalten. Diese sind jedoch sehr teuer und nur für größere Serversysteme benutzbar. *Tools* wie metal & CSSV können Fehler statisch erkennen, Purify & SafeMem sogar dynamisch zur Laufzeit.

2.2. Disk Corruption

Disk Corruption liegt vor, wenn Daten auf einem Datenträger einem nicht erwarteten Zustand besitzen. Das Problem befindet sich also auf dem Speicherstack.

Ursachen für Disk Corruption können das Altern der Magnetschicht eines Datenträgers oder physikalische Beschädigungen wie Kratzer auf dem Medium sein. Außerdem können Bugs in der Laufwerkfirmware, in Gerätetreibern oder im Betriebssystem Fehler generieren. Daten können fälschlicherweise an einen Ort geschrieben werden, der nicht als Zielort angegeben wurde oder Phantom-Writes können auftreten. Bei diesen wird eine Meldung ausgegeben, dass der Schreibvorgang erfolgreich, obwohl keine Daten geschrieben wurden.

Zur Fehlerbehebung nutzt man Prüfsummen, welche bereits im Abschnitt 1.2. erläutert wurden. Durch Redundanz ist das Entdecken und teilweise Beheben von Fehlern möglich. Es werden Kopien von Superblöcken erstellt, die redundante Zeiger auf einen Block inne haben. Schließlich werden bei RAID-Systemen mehrere physische Festplatten zu einem logischen Laufwerk verknüpft. Dadurch erhält man einen höheren Durchsatz bei eventuellen Ausfällen einzelner Platten. Ein Nachteil von RAID-Systemen ist, dass einzelne fehlerhafte Blöcke toleriert werden, welche zu im schlimmsten Fall zu ganzen beschädigten Dateien führen können.

2.3. CERN-Studie

Um abzuschätzen, wie hoch die Wahrscheinlichkeit für das Auftreten von Memory und Disk Corruption ist und wie groß das Ausmaß der dadurch verursachten Schäden ist, führte das CERN Institut eine Studie durch, in der 15 Petabyte (entspricht 15.000 Terabyte) an Daten untersucht wurden. Diese wurden durch hochenergetische Teilchen innerhalb kurzer Zeit erzeugt und mehrere Monate auf silent data corruption, also unbemerkter Datenbeschädigung, überprüft. Die Studie ist in 3 Ebenen untergliedert: Disk Error, RAID-Error und Memory Error.

Bei den *Disk Errors* wurde eine spezielle 2 Gigabyte große Datei alle 2 Stunden auf einen Datenträger der über 3000 vorhandenen Knotenpunkte geschrieben und anschließend auf Fehler überprüft, indem die beschriebenen Abschnitte ausgelesen wurden. In 5 Wochen wurden 500 Fehler an 100 Knotenpunkten gefunden. Davon machten betroffene 512 Byte Sektoren und einzelne Bits jeweils schätzungsweise 10% der Fehler aus. 80% waren durch fehlerhafte 64 KB Regionen bedingt, was im Endeffekt auf einen Bug in der Firmware zurückzuführen war. Nachdem ein entsprechendes Update durchgeführt wurde, trat dieser Fehler nicht mehr auf.

4 Wochen lang führte man auf 492 *RAID-Systemen* den Verify-Command aus, bei dem auf Fehler geprüft wird, wenn Schreibvorgänge stattfinden. Eine Fehlerrate von 10^-14 wurde eingangs vorhergesagt, die tatsächliche Fehlerrate betrug schließlich nur ein Drittel der Schätzung. Man fand bei einer Datenmenge von 2,4 Petabyte insgesamt 300 Fehler. Umgerechnet entspricht dies einem Fehler auf 11,37 Terabyte.

Memory Errors betreffend gingen 3 Double-Bit Errors auf 1300 Knotenpunkten nach 3 monatiger Beobachtungen hervor. Obwohl dies eine recht kleine Zahl ist, ist dieses Resultat ziemlich problematisch. Man ist ursprünglich davon ausgegangen, dass es keine Fehler geben dürfen. Hinzu kommt, dass Double-Bit Errors nicht korrigierbar sind.

Abschließend wurden 8,7 Terabyte an Nutzerdaten bzw. 34 000 Daten untersucht. 22 dieser 34 000 Daten wiesen Beschädigungen auf, was einer Fehlerrate von 1 zu 1500 entspricht. Die Fehlerrate nach der gesamten Studie betrug 2x10^-7, welche deutlich höher ist als die erwarteten 10^-14. Diese Zahl ist weniger schlimm, fatal ist jedoch, dass die Fehlerrate in den einzelnen Bestandteilen wie CPU und Datenträger sehr hoch war.

Heutzutage befinden sich auf einer Festplatte mit einem Terabyte Datenkapazität ca. 3 fehlerhafte Dateien, die ihre Ursache in silent data corruption haben. Als Zukunftsprognose lässt sich daraus ableiten, dass bei steigenden Kapazitäten von Datenträgern eine steigende Anzahl beschädigter Dateien mit einhergeht.

3. ZFS

Das Zettabyte File System ist ein von Sun Microsystems entwickeltes 128 Bit-Dateisystem. Das Dateisystem bringt vor allem auf Datenintegrität bezogen ein breit gefächertes Spektrum an Funktionalitäten mit sich. Kernidee der Entwickler von ZFS war, ein Dateisystem zu schaffen, das unendlich große Datenmengen verwalten. Darauf soll bereits der Name aufmerksam machen; Ein Zetabyte entspricht 10^{21} Byte. Im Vergleich: Ein Terabyte sind lediglich 10^{12} Byte. Im Folgenden werden die Merkmale von ZFS näher erläutert.

3.1. Copy-On-Write

Beim *Copy-On-Write-Verfahren* (zu deutsch: Kopieren-Beim-Schreiben) werden Datenblöcke nicht überschrieben sondern zunächst vollständig an einen freien Platz geschrieben. Danach werden die Verweise in den Metadaten aktualisiert und die alten Datenblöcke gelöscht. Das bedeutet, dass die jeweiligen Metadaten nie gelöscht werden, sondern stattdessen dem jeweiligen Speicherabbild zugeordnet werden. Ziel dieses Verfahrens ist es, unnötige Kopien und Schreibvorgänge zu vermeiden.

3.2. Snapshots

In ZFS können Snapshots des Systems erstellt werden. Bei diesen wird der aktuelle Systemzustand gespeichert.

3.3. Software-RAID:

ZFS bringt ein eigenes Software-RAID-System mit sich. In diesem sind Pools über RAID-System ausfallgesichtert. Hierbei werden aus den physischen Datenträgern Redundanzgruppen gebildet. Im Software-RAID finden sich folgende Stufen: Beim *RAID-1*, der Spiegelung, speichern 2 oder mehr Festplatten alle Daten mindestens 2 mal. Das *RAID-Z* bildet aus mindestens 3 Festplatten eine paritätsgesicherte Redundanzgruppe. Dadurch kommt es bei eventuellen Festplattenausfall zu keinem Datenverlust. Paritätsgesichert bedeutet, dass an Bitfolgen die un/gerade Anzahl an Einsen betrachtet wird. Es werden entweder alle Bitfolgen gerade oder alle ungerade gemacht, indem ein sogenanntes Kontrollbit angefügt wird. Nachteile der Paritätssicherung sind, dass bei diesem Verfahren keine Fehlerkorrektur stattfindet und zudem nur eine ungerade Anzahl an Fehlern erkennbar ist. Der Unterschied zwischen dem speziellen in ZFS integrierten RAID-Z im Gegensatz zum klassischen RAID-5 ist, dass es nicht zu *„Write Holes“* kommen kann. „Write Holes“ bezeichnet man als das Phänomen, dass Daten auf einen Datenträger geschrieben wurden, jedoch die Paritätsinformation nicht mitgeschrieben wurden. Bei einem Plattenausfall würde es zu einer fehlerhaften Wiederherstellung der Daten kommen, was einen partiellen Datenverlust bedeutet. Ein weiterer großer Vorteil des RAID-Systems in ZFS ist, dass zwischen freien und belegten Datenblöcke unterschieden wird. Bei einer RAID-Rekonstruktion würden nämlich nur belegte Datenblöcke gespiegelt werden. Im Schadensfall gewinnt der Benutzer dadurch eine enorme Zeitersparnis.

3.4. prüfsummenbasierter Schutz

Jede logische Einheit in ZFS bildet einer binären Hash-Baum. Somit erhält jeder einzelne Datenblock eine Prüfsumme. Im übergeordneten Block entsteht ein neuer Wert, der aus den Werten des untergeordneten und seines benachbarten Blocks zusammengesetzt ist. Deshalb werden bei diesem Verfahren mehr Fehler erkannt als bei normalen Prüfsummen. Wenn nämlich überschriebene Blocks erkannt wurden, so können auch darüber liegende Hash-Werte nicht mehr richtig sein.
Der Prüfsummenschutz in ZFS erkennt Fehler automatisch sofort und behebt diese, ohne dass ein manueller Eingriff vonnöten. Der dadurch entstehende Performance-Verlust ist hierbei minimal gehalten, der Benutzer ist davon deshalb nicht betroffen. In hohem Maße vorteilhaft ist, dass sich das Dateisystem dadurch ständig in einem konsistenten Zustand befindet. Eine Überprüfung der Datenträger nach einem Stromausfall bleibt dem Benutzer folglich erspart.

3.5. Eigenes Fazit

ZFS ist ein sehr vielseitiges Dateisystem, welches diverse Funktionalitäten mit sich bringt, welche insbesondere im Bezug auf Datenintegrität noch von keinem anderen Dateisystem geleistet werden. Hervorzuheben ist, dass das Dateisystem Datenintegrität aktiv sicherstellt, womit dem Benutzer lästige Arbeit erspart wird. Ehemals für Server- und Rechenzentren gedacht, in denen große Datenmengen und Datenintegrität eine bedeutsame Rolle spielen, entwickelte sich das Dateisystem ZFS aufgrund steigender Leistungen heutiger Rechnersysteme ebenfalls zur Option für Arbeitsplatzrechner. Die früher aufwendig zu bearbeitenden 128-Zeiger stellen für heutige Systeme kein Hindernis mehr dar.