Prüfungsprotokoll Verlässliche Verteilte Systeme

Patrick Elftmann

14. Oktober 2005

Fächer: Verlässliche Verteilte Systeme 1+2 (V4/V3), Datenkommunikation (V3), Mobilkommunikation (V2)
Prüfer: Prof. Freiling (geb. Gärtner)
Dauer: 35 Minuten
Note: 1,0

In der Regel fragt Professor Freiling, im Folgenden FF, am Anfang der Prüfung mit welchem Themengebiet man anfangen möchte. Dies erleichtert einem den Einstieg und nimmt die Nervosität. Doch es kann auch anders kommen...

Verlässliche Verteilte Systeme 2 (20 Minuten)

FF: Hast Du Dir schon überlegt womit Du anfangen möchtest?

PE: Ich würde gerne mit CHT anfangen

FF: Also mit VVS2. Und dann?

PE: Dann würde ich gerne mit VVS1 weitermachen, aber ich bin mir sicher, dass wir dann schnell auf die Datenkommunikation und Mobilkommunikation zu sprechen kommen

FF: Also VVS2 und dann ein Konglomerat aus dem Rest. Gut.
CHT scheint bei den Studenten sehr beliebt zu seinen, aber sei mir nicht böse, wenn ich nicht den kompletten Beweis hören möchte. CHT habe ich heute schon 3x gehört. Ich werde einfach nur ein paar Fragen dazu stellen. Aber bevor wir in den Beweis einsteigen, vorher noch ein paar allgemeine Fragen zu Consensus: Was ist Consensus und worum geht es da?

PE: Beim Consensus schlägt jeder Prozess einen Wert vor und wie der Name schon sagt, gilt es nun einen Konsens zu finden, d.h. die Prozesse müssen sich auf einen Wert einigen. Dabei muss jeder Consensus-Algorithmus die folgenden 4 Eigenschaften erfüllen:

1. *C1. Validity*: Any value decided is a value proposed (Safety)
2. *C2. Agreement*: No two correct processes decide differently (Safety)
3. *C3. Termination*: Every correct process eventually decides (Liveness)
4. *C4. Integrity*: No process decides twice (Safety)

FF: Consensus ist ja doch ein schwierigeres Problem. Warum?

PE: Im Vergleich zu Reliabe Broadcast braucht man beim Consensus zum Beispiel einen Fehlerdetektor. Man braucht also ein Stück Synchronität. Consensus ist also schwieriger als Reliable Broadcast.

FF: (*schaut noch nicht überzeugt*)

PE: FLP beweist, dass es keinen deterministischen Consensus-Algorithmus in einem asynchronen System gibt, der einen oder mehrere Crashes toleriert. Man braucht also einen Fehlerdetektor. In dem Sinne ist Consensus ein schwieriges Problem.

FF: Ja genau! Wie zeigt man denn, dass ein Fehlerdetektor der Schwächste für ein Problem ist?

PE: Ein Fehlerdetektor D ist der Schwächste für einen Abstraktion A wenn

1. Der Fehlerdetektor D ist hinreichend, d.h. D implementiert A
2. Der Fehlerdetektor ist notwendig, d.h. jeder andere Fehlerdetektor, der A implementiert implementiert auch D

CHT ist also der notwendige Teil des Beweises, dass Omega der schwächste Fehlerdetektor für Consensus mit einer Mehrheit an korrekten Prozessen ist

FF: Wie wird das denn beim CHT-Beweis gemacht?

PE: Wir konstruieren einen Algorithmus T, der letztendlich Omega implementiert. T benutzt dabei einen Consensus-Algorithmus A und seinen Fehlerdetektor D.
Dabei liegt die Schwierigkeit darin, dass wir nicht wissen, wie A funktioniert und welche Informationen uns der Fehlerdetektor D liefert.

FF: Warum ist dies beim Consensus im Vergleich zu anderen Problemen so schwierig zu zeigen?

PE: Beim TRB oder NBAC werden Fehlerannahmen explizit in den Eigenschaften erwähnt. Dies ist beim Consensus nicht der Fall.

FF: Wie extrahiert man beim CHT den korrekten Prozess? Wie ist die Idee?

PE: Jeder Prozess simuliert lokal den Conesensus-Algorithmus A für alle anderen Prozesse. Aus dieser Simulation heraus kann man einen korrekten Prozess extrahieren, dem dann alle Prozesse vertrauen. Somit wäre Omega, der einen „eventually unique leader“ ausgibt, implementiert

FF: Wieso stellen alle Pfade des DAG eine gültige Aktivierungsreihenfolge der Prozesse dar?

PE: Wir befinden uns in einem asynchronem System, d.h. Laufzeiten von Berechnungen und Nachrichtenversand sind nicht beschränkt. Der DAG beinhaltet eine reale Fehlerdetektorausgabenfolge, d.h. die Fehlerdetektorausgaben könnten bei einem realen Consensuslauf wirklich so gelesen worden sein. Die Fehlerdetektorinformationen dürfen aber auch „überlesen“ werden, welches durch die transitive Abgeschlossenheit des DAGs ausgedrückt wird. Der Grund hierfür liegt wiederum in der unbeschränkten Zeit für einen Ausführungszeit des Prozesses.

FF: Nehmen wir an, der kritische Index ist stabilisiert. Wie extrahiert man den korrekten Prozess?

PE: Die Konfigurationen c_0 bis c_{k-1} sind 0-valent. Wir nehmen an c_k ist 1-valent. Dann ist der Prozess p_k korrekt und alle anderen Prozess vertrauen ihm.
Natürlich muss man auch zeigen, dass wenn c_k bivalent ist, ein korrekter Prozess extrahierbar ist. Dies wird im Anhang des Papers gezeigt und würde den zeitlichen Rahmen hier sprengen. Deshalb beschränken wir uns hier auf den Fall, dass c_k 1-valent ist.
Beweis durch Widerspruch. Wir nehmen also an, dass p_k faulty ist. Dann nimmt p_k irgendwann am Aufbau des DAG nicht mehr teil. D.h. es gibt einen Pfad, auf dem p_k keine Schritte ausführt, wegen der transitiven Abgeschlossenheit des DAG. Wenn wir uns nun die Konfigurationen c_{k-1} und c_k anschauen, unterscheiden sich diese nur in dem vorgeschlagenen Wert von Prozess p_k. Ein anderer korrekter Prozess p_i kann nun aber einen Consensus-Lauf ausgehend von c_{k-1} und einen Lauf ausgehend von c_k nicht unterscheiden. Also muss auch bei einer Simulation ausgehend von c_k mal 0 entschieden werden, da ja c_{k-1} 0-valent ist. Dies ist aber ein Widerspruch dazu, dass c_k 1-valent ist. Und insgesamt also ein Widerspruch zur Annahme. p_k muss also korrekt sein

FF: Wo werden die Eigenschaften von Consensus im CHT benutzt?

PE: Die *Agreement*-Eigenschaft wird explizit im Anhang benötigt um zu zeigen, dass ein korrekter Prozess extrahierbar ist, wenn c_k bivalent.
Aus der *Validity* folgt die 0-Valenz von c_1 und die 1-Valnez von c_n.
Die *Termination* von Consensus wird benötigt um zu taggen. Denn wenn Consensus nicht mit einer Entscheidung terminieren würde, könnte wir die Konfigurationen nicht taggen.
Integrity wird explizit nicht benötigt, aber ohne die Integrity könnte man die Konfigurationen wiederum nicht taggen

FF: Es gibt viele verschiedene Fehlerdetektoren. Einige sind auch äquivalnet, wie zum Beispiel Omega und $\diamond$S. Wie zeigt man das?

PE: Um $\diamond$S durch Omega zu implementieren fügt man alle Prozesse einer Variable Emulate hinzu und entfernt den „eventually unique leader“, den Omega ausgibt. Dies simuliert die Strong Completness und die Eventually Weak Accuracy von $\diamond$S.
Um Omega durch $\diamond$S zu implementieren besitzt jeder Prozess für jeden anderen Prozess einen Counter. Dieser wird bei einer Verdächtigung durch den Fehlerdektor erhöht. Die Counter werden periodisch an alle andere Prozesse gebroadcastest und jeder Prozess setzt seine Counter auf das Maximum der empfangenen Counter und der Eigenen. Der Prozess mit dem niedrigsten Counter wird ausgegeben.

FF: Und was ist, wenn zwei Prozesse die gleiche Counterzahl haben?

PE: Das ist egal. Dann kann man zum Beispiel den Prozess ausgeben, der den kleinsten Index hat.

FF: Die Fehlerdetektoren sind also äquivalent. Allerdings sind die Fehlerdetektoren unter anderen Gesichtspunkten nicht mehr identisch. Zum Beispiel im Bezug auf den benötigten Speicherplatz. Einige Fehlerdetektoren brauchen nur begrenzten Speicherplatz, andere unbegrenzten. In Bezug auf den Speicherplatz, welcher ist schwächer? Omega oder $\diamond$S?

PE: ???

FF: Wenn man $\diamond$S durch Omega implementiert, wieviel Speicherplatz braucht man da?

PE: Man braucht nur endlich viel Speicher. Nur soviel Speicher, wie man für die Variable Emulate braucht.
Wenn man Omega duch $\diamond$S implementiert braucht man aber unbegrenzten Speicher. Der Speicherplatz für die Counter wächst unendlich.

FF: Also welcher Fehlerdetektor ist dann schwächer?

PE: $\diamond$S ist schwächer, weil Omega den Fehlerdetektor $\diamond$S mit begrenztem Speicherplatz implementiert. Umgekehrt ist dies aber nicht möglich.

FF: Wie kann man in einem Partial Synchronous System einen Eventually Perfect Fehlerdetektor bauen?

PE: ???

FF: Wie implementiert man denn einen perfekten Fehlerdetektor in einem synchronen System?

PE: In einem sychronen System sind obere Schranken für den Nachrichtenversand bekannt. Also kann man hier mit Timeouts arbeiten. Wenn ein Timeout verstrichen ist, muss also der Prozess abgestürzt sein.

FF: Und wie baut man einen Eventually Perfect Fehlerdetektor in einem synchronen System?

PE: Auch hier arbeitet man mit Timeouts. Nach Ablauf des Timeouts verdächtigt man den Prozess. Falls dann doch noch eine Nachricht von einem verdächtigten Prozess ankommt, vergrößert man den Timeout

FF: Was wär denn ein geeignetes Maß für ein Timeout?

PE: In der Datenkommunikation wäre es standardmäßig die Round-Trip-Time (RTT). Die könnte man hier auch nehmen

FF: Noch mal zurück zu der Frage wie ich einen Eventually Perfect Fehlerdetektor in einem Partial Synchronous System bauen kann

PE: Gar nicht, denn ich weiss ja nicht wann die Synchronität einsetzt

Verlässliche Verteilte Systeme 1
Datenkommunikation (10 Minuten)

FF: Überleitung zu VVS1 (*kann mich gar nicht mehr an der Wortlaut erinnern*)
Erklär doch mal IP-Failover

PE: (*erklärt*) ...

FF: Du sagtest dass viele Router standardmäßig Gratiutous-ARP wegen ARP-Spoofing blocken. Warum machen die Router das? Wie funktioniert ARP-Spoofing?

PE: ARP-Spoofing bezeichnet das Senden von gefälschten ARP-Paketen.
Beim ARP-Spoofing wird das gezielte Senden von gefälschten ARP-Paketen dazu benutzt, um den Datenverkehr zwischen zwei Hosts in einem Computernetz abzuhören oder zu manipulieren. Man spricht dabei von einem Man-In-The-Middle-Angriff. Um den Datenverkehr zwischen Host A und Host B abzuhören, sendet der Angreifer an Host A eine manipulierte ARP-Nachricht, so dass dieser Pakete für Host B an den Angreifer sendet. Das selbe geschieht mit Host B, so dass dieser Pakete statt direkt an A nun ungewollt zum Angreifer sendet. Der Angreifer muss nun die von A und B erhaltenen Pakete an den eigentlichen Empfänger weiterleiten, damit eine abhörbare Verbindung zustande kommen kann

FF: Kennst Du Programme mit denen man ARP-Spoofing, also eine MITM-Attacke, betreiben kann?

PE: Ich habe diverse Programme im Hacker-Praktikum kennengelernt. Die gebäuchlichsten dürften „Ettercrap“ für Linux und „Cain & Abel“ für Windows sein

FF: Angenommen ich arbeite per Telnet. Welcher Angriff ist da möglich?

PE: TCP-Hijacking der aktuellen Session, d.h. abhängen des Clients durch ein RTS und einschleusen von Kommandos/Daten zum Server. „Take-No-Prisoner Angriff“

FF: Wieso kann man Sequenznummern erraten?

PE: Die Sequenznummern zählen die übertragenden Bytes. Sie werden also stets erhöht.

FF: Ist dieser Angriff bei SSH immer noch möglich?

PE: Ja, das Abhängen des Clients ist durch erraten er richtigen Sequenznummer und senden eines TCP-Packets mit gesetzter RST-Flag immer noch möglich. Allerdings können keine Kommandos in die Session eingeschleust werden.

FF: Warum ist dies nicht möglich? Auf welcher Ebene würde man ein gescheitertes Hijacking erkennen? Auf IP- oder TCP-Ebene?

PE: Das Einschleusen von eigenen Kommandos/Daten ist nicht möglich, da dem Angreifer der Session-Key zur Verschlüssung der Daten unbekannt ist.
Der Versuch die Verbindung zu übernehmen wird weder auf IP- noch auf der TCP-Ebene bemerkt. Hier ist ja anscheinend alles in Ordnung, die Sequenz- und Bestätigungsnummer sind korrekt. Allein die SSH-Applikation würde einen Einbruchversuch bemerken

FF: IP-Pakete haben doch noch ein Identifikationsfeld. Kann man dort nicht noch eine Unterscheidung der IP-Pakete treffen um das Erraten der Sequenznummer zu erschweren?

PE: IP-Pakete die fragmentiert werden, besitzen dieselbe IP-ID. Allerdings setzen viele Router das „Don't Fragment"-Flag und setzten die IP-ID immer auf Null. Mit der IP-ID lassen sich also IP-Pakete nicht unbedingt weiter unterscheiden. Dies macht es auch schwierig Zombie-Rechner für das Idle-Scanning zu finden.

FF: Idle-Scanning! Ich sehe, da kennst Du Dich auch aus. Dann brauch ich das ja nicht mehr zu fragen

Mobilkommunikation (5 Minuten)

FF: Gehen wir doch mal über zur Mobilkommunikation. Wie funktioniert Mobile-IP?

PE: Ein mobiles IP muss vier Anforderungen erfüllen:

1. Transparenz: Mobile Endgeräte behalten ihre IP-Adresse und die Wiederaufnahme der Kommunikation nach Abtrennung muss möglich sein
2. Kompatibilität: Unterstützung der gleichen Schicht-2-Protokolle wie IP und keine Änderungen an bisherigen Rechnern und Routern
3. Sicherheit: Alle Registrierungsnachrichten müssen authentifiziert werden
4. Effizienz und Skalierbarkeit: Möglichst wenige zusätzliche Daten zum mobilen Endgerät

Mobile-IP funktioniert dabei wie folgt:

- Sender sendet an IP-Adresse von MN, HA fängt Paket ab
- HA tunnelt Paket an COA, hier FA, durch Kapselung
- FA leitet das Paket an MN weiter

FF: Funktioniert SSH auch zusammen mit Mobile-IP?

PE: Mir fällt kein Grund ein, warum es nicht gehen sollt. Ich denke es funktionert also.

FF: Ich denke auch, dass es funktioniert. Habe es aber noch nie ausprobiert.
Dann denke ich, machen wir hier jetzt mal Schluss...

PE: (*schaue verdutzt auf die Uhr*) Schon?

FF: Du kannst ja eh alles. Da könnten wir Dich noch lange prüfen ...

Fazit

Professor Freiling ist der angenehmste Prüfer, den ich bisher hatte. Er schafft es durch seine ruhige und zum Teil witzige Art eine entspannte Prüfungsatmosphäre zu schaffen. Die Prüfung ist kein Frage-Antwort-Spiel sondern ein Gespräch welches sich permanent weiterentwickelt. Wenn man also geschickt ist, kann man so die Prüfung teilweise leiten und den Verlauf mitbestimmen.
Versteht man mal auf Anhieb nicht sofort die Frage, stellt er sie anders oder leitet einen mit einen anderen Frage zum richtigen Ergebnis. Wurde der Prüfungsstoff gut gelernt und die Prüfung gut vorbereitet, kann eigentlich nichts schief gehen.
Viel Erfolg!