Practical Byzantine Fault Tolerance
Ein Algorithmus von Miguel Castro

Christian Kellermann
Christian.Kellermann@stud.uni-erlangen.de

Kurzzusammenfassung

Hier soll der Practical Byzantine Fault Algorithmus von Miguel Castro vorgestellt werden. Der Algorithmus ermöglicht die Replikation von Diensten in einem Netzwerk, das bis zu $\frac{n-1}{3}$ fehlerhafte Knoten enthalten kann. Die Toleranz gegenüber byzantinischen Fehlern, sowie proactive recovery Methoden sind die Stärken dieses Algorithmus.

1 Einführung

Der BFT Algorithmus ermöglicht die Implementierung eines Dienstes, der unter bestimmten Voraussetzungen byzantinische Fehler tolerieren kann. Der Dienst wird von n Knoten (Replikaten) angeboten. Das Ergebnis einer Anfrage wird durch Einigung der Knoten ermittelt.

2 BFT-PK: BFT mit signierten Nachrichten

2.1 Grundsätzliche Annahmen

Alle n Replikate bieten Dienste an, die durch endliche, deterministische Automaten repräsentierbar sind. Die Knoten, sowie der Client sind miteinander durch ein unzuverlässiges Netzwerk verbunden. Die Übermittlung der Nachrichten erfolgt per multicast. Eine Nachricht m eines Replikats wird durch eine Signatur authentifiziert. Es wird angenommen, daß die Signatur einer Nachricht nicht gefälscht werden kann. Zusätzlich verwendet der Algorithmus eine Hash-Funktion, um Prüfsummen von Nachrichten zu ermitteln. Diese Hash-Funktion soll keine Kollisionen aufweisen. Fehlerhafte Knoten sollen sich beliebig verhalten dürfen. Die Knoten müssen zu Beginn bekannt sein und darf sich im laufenden Algorithmus nicht verändern.

Eines der Replikate übernimmt die Aufgabe des primary, alle restlichen Replikas sind backups. Der Zustand einer primary-backup Konfiguration, d.h. welcher Knoten primary ist, wird als view bezeichnet. Views werden laufend nummeriert, der primary Knoten p einer Sicht v ergibt sich aus: $p = v \bmod |R|$, mit R als der Menge der Replikate.

Der primary Knoten bestimmt die Reihenfolge der Ausführung von Anfragen. Jeder Anfrage wird eine fortlaufende Sequenznummer zugewiesen, an Hand derer eine Anfrage eindeutig im System bestimmt ist. Natürlich könnte ein Ausfall des primaries zu falschen Sequenznummern führen oder der primary Knoten kann sich anderweitig falsch verhalten. Die backup Knoten reagieren auf falsche Sequenznummern und wählen einen neuen primary Knoten (siehe unten).

Der Algorithmus geht davon aus, dass clients erst eine neue Anfrage stellen, wenn die Vorherige abgearbeitet wurde.

2.2 Quoren und Zertifikate

BFT-PK verläßt sich auf Quoren um die Richtigkeit verschiedener Phasen des Algorithmus sicherzustellen. Ein Quorum besteht hierbei aus $2f + 1$ Replikaten, mit f als Anzahl fehlerhafter Replikate. Da es insgesamt $3f + 1$ Replikate geben soll, folgt:

- Zwei Quoren haben mindestens ein korrektes Replikat gemeinsam und
- Es gibt immer ein Quorum, das nur aus korrekten Replikas besteht.

Jedes Mitglied eines Quorums sammelt Empfangsbestätigungen von Nachrichten von anderen Quorumsmitglieder. Diese werden Zertifikate genannt, genauer Quorums-Zertifikate (quorum certificates). Neben diesen verwendet BFT-PK auch so genannte schwache Zertifikate, bei denen es wenigstens $1 + f$ Bestätigungen von anderen Replikaten sein müssen. Die Schwachen Zertifikate beweisen dass wenigstens ein korrekt funktionierendes Replikat die Nachricht erhalten hat.

2.3 Normaler Ablauf

Wir gehen zunächst von einem System aus, dass nur funktionierende Knoten enthält. In diesem wird eine Anfrage eines clients c nach dem Ergebnis einer Operation o durch senden einer $\langle REQUEST, o, t, c\rangle_{\delta_c}$ Nachricht an den primary Knoten. Der Parameter t ist ein Zeitstempel und wird benötigt um eine exactly once Semantik zu ermöglichen. Jedes Replikat antwortet mit einer $\langle REPLY, v, t, c, i, r\rangle_{\delta_i}$ Nachricht an den client. Als Parameter wird die aktuelle Sicht v, der Zeitstempel des Requests, die Nummer i des Replikats und das Ergebnis r der Operation angegeben.

Der client warten, bis er ein schwaches Zertifikat, mit gleichem Zeitstempel t und Ergebnis r erhält und akzeptiert dann das Resultat als richtig. Dieses Zertifikat wird als reply certificate bezeichnet.

Wenn das reply certificate nicht rechtzeitig erreicht wird, versendet der Client den request erneut, diesmal an alle Knoten. Wenn die Anfrage bereits abgearbeitet wurde, schicken die Replikate ihren reply erneut.(Replikate speichern jeweils die letzte reply Nachricht die sie einem client gesckickt haben.) Andernfalls wird der request an den primary Knoten weitergeleitet. Sollte dieser die Anfrage nicht an das System weiterleiten wird ein neuer primary Knoten ausgewählt.

2.3.1 3 Phasen Commit

Der primary Knoten schickt eine Anfrage des clients mittels eines 3-Phases-Commit Protokolls an die backup Knoten. Das Protokoll besteht aus folgenden Phasen: pre-prepare, prepare und commit. Wenn der primary eine Anfrage m erhält, erzeugt er für diese eine Sequenznummer n und verschickt ein $\langle PRE - PREPARE, v, n, m\rangle_{\delta_p}$ an alle backup Knoten, wobei v die aktuelle Sicht angibt. Die $PREPARE$-, wie auch die $COMMIT$-Nachricht enthält ebenfalls die aktuelle Sicht des primary Knotens.

Backup Knoten akzeptieren nur Nachrichten die aus derselben Sicht entstanden sind, wie ihre eigene und die Sequenznummer inner halb einer unteren und einer oberen Schranke h bzw H liegt. Diese so genannten low- bzw high- watermarks verhindern den Mißbrauch der Sequenznummer durch den primary Knoten, d.h. Sequenznummern können nicht aufgebraucht werden, indem sehr hohe Nummern gewählt werden, bzw. nur alte Nummern gewählt werden. Die Schranken spielen auch bei der garbage collection eine Rolle.

Ein backup i akzeptiert eine pre-prepare Nachricht, wenn es noch keine pre-prepare Nachricht für eine Sequenznummer n in der Sicht v akzeptiert hat. Wenn i die Nachricht akzeptiert,

schickt es eine $\langle PREPARE, v, n, d, i \rangle_{\delta_i}$ Nachricht mit der Sicht v, der Sequenznummer n und der gehashten Nachricht d (ein so genannter digest) an alle Replikate. In jedem anderen Fall wird die eingegangene Nachricht ignoriert. Mit dem Senden einer $PREPARE$-Nachricht gibt ein Replikat zu erkennen, dass es der Zuorndung Nachricht - Sequenznummer zustimmt.

Anschliessend sammelt jedes Replikat $REQUEST$-Nachrichten bis es ein Quorums-Zertifikat für die jeweilige Sequenznummer, Sicht und Request empfangen hat. Danach haben sich alle Replikate auf die Abfolge der Requests geeinigt. Erlangt ein Replikat ein Quorums-Zertifikat so sendet es eine $\langle COMMIT, v, n, d, i \rangle_{\delta_i}$ Nachricht an das System.

Nun wartet der Knoten bis er ein $COMMIT$-Zertifikat erhalten hat. Danach wird die Anfrage ausgeführt, sofern keine früheren Anfragen zur Ausführung anstehen. Das Ergebnis wird wie oben erwähnt mit einer $REPLY$ Nachricht an den client gesandt.

2.3.2 Logging

Um ein Einhalten der Reihenfolge zu gewährleisten und um der Möglichkeit Rechnung zu tragen, dass Nachrichten zeitlich ungeordnet eintreffen können, werden logs benötigt. Jeder Knoten protokolliert die jeweiligen $PRE-PREPARED$, $PREPARED$ und $COMMIT$ Nachrichten die er selbst verschickt hat, sowie die eingegangenen $REQUEST$ und die versendeten $REPLY$ Nachrichten. Es ist ausreichend wenn flüchtige Speicher für logs verwendet werden.

2.4 Garbage Collection

Um ein stetiges Anwachsen der logs zu verhindern muss ein Garbage Collection Mechanismus implementiert werden. Allerdings können nicht alle Nachrichten eines ausgeführten Requests ohne weiteres aus dem Log gelöscht werden. Informationen zu einem erlangten Zertifikat könnten im weiteren Verlauf noch benötigt werden. Ein Knoten müsste also vor jedem Löschvorgang verifizieren dass sein Status richtig ist. Eine solche Verifikation ist jedoch zu aufwändig um sie nach jedem $COMMIT$ durchzuführen. Daher wird ein solcher Abgleich periodisch durchgeführt und zwar dann, wenn die Sequenznummer durch die checkpoint period K teilbar ist. Ein solcher Abgleich wird als checkpoint bezeichnet, ein bestätigter Abgleich, d.h. gültig im gesamten System, wird als stable chekcpoint bezeichnet.

Ein Replikat erzeugt einen checkpoint, indem es eine $\langle CHECKPOINT, v, n, d, i \rangle_{\delta_i}$ Nachricht verschickt, wobei hier n die Sequenznummer, des zuletzt ausgeführten Requests und d der Digest dieses Requests ist. Es genügt ein schwaches Zertifikat, um einen checkpoint zu bestätigen. Sobald ein Replikat dieses so genannte stable certificate erhalten hat, kann es alle Nachrichten mit einer $Sequenznummer <= n$ aus dem log entfernen.

Das checkpoint Protokoll wird auch zum Setzen der water marks benutzt. Dabei ist h die Sequenznummer des letzten stable checkpoint und $H = h + L$, wobei L die Größe des Logs darstellt. L bestimmt die Anzahl der aufeinander folgenden Sequenznummern die ein Replikat in das Log aufnimmt und errechnet sich aus $L = \lambda * K$, mit λ als ein kleiner, ganzzahliger Faktor (z.B. 2). Dadurch wird es unwahrscheinlich, dass ein Replikat vergebens auf einen stable checkpoint wartet.

2.4.1 View Changes

Um deadlocks zu vermeiden, wenn der primary knoten ausfällt, gibt es eine Möglichkeit für backup Knoten einen neuen primary zu wählen. Das geschieht durch eine Veränderung der aktuellen Sicht *viewchange*. View changes werden durch time outs ausgelöst. Jedes backup

Replikat startet einen Timer beim Erhalt eines Requests. Wenn ein Request abgearbeitet wurde wird der Timer bei eintreffen eines neuen Requests wieder aufgezogen.

Wenn der Timer des backups i in der Sicht v abläuft, startet backup i einen view change nach Sicht $v+1$. Es werden keine Nachrichten mehr angenommen (außer checkpoint, view-change und new-view Nachrichten) und eine $\langle VIEW-CHANGE, v+1, n, s, \mathcal{C}, \mathcal{P}, i\rangle_{\delta_i}$ Nachricht wird an alle Replikate gesendet. In diesem Fall ist n die Sequenznummer des letzten stable checkpoints s, $\mathcal{C}$ ist das Zertifikat für den checkpoint und $\mathcal{P}$ ist eine Menge mit prepared Zertifikaten für Requests mit einer Sequenznummer größer als n.

Hat der neue Primary ein Quorum-Zertifikat für die $VIEW-CHANGE$ Nachricht erhalten, sendet er eine $\langle NEW-VIEW, v+1, \mathcal{V}, \mathcal{O}, \mathcal{N}\rangle_{\delta_p}$ Nachricht an alle Knoten. Hier ist $\mathcal{V}$ das new view Zertifikat und $\mathcal{O} \cup \mathcal{N}$ eine Menge von pre-prepare Nachrichten, die die Sequenznummern für die neue Sicht bekannt geben. Diese werden wie folgt berechnet:

1. Der primary Knoten bestimmt die kleinste Sequenznummer h des letzten stable checkpoints in $\mathcal{V}$, sowie die höchste Sequenznummer H in $\mathcal{V}$
2. Der primary erzeugt pre-prepare Nachrichten für die Sicht $v+1$ für jedes n $h < n \leq H$ Hier gibt es zwei Fälle:
   (a) es exisiert ein prepared Zertifikat in $\mathcal{V}$ mit der Sequenznummer n oder
   (b) es existiert kein solches Zertifikat. Im ersten Fall wird eine $\langle PRE-PREPARE, v+1, n, m\rangle_{\delta_p}$ Nachricht zu $\mathcal{O}$ hinzugefügt. Im zweiten Fall wird eine $\langle PRE-PREPARE, v+1, n, null\rangle_{\delta_p}$ zu $\mathcal{N}$ hinzugefügt. Ein null-digest bewirkt, dass clients in der commit-phase eine NOP durchführen.

Als nächstes legt der primary $\mathcal{O}$ und $\mathcal{N}$ im log ab, sowie das stable certificate für den letzten stable checkpoint falls nötig. Erst jetzt wechselt der primary in die Sicht $v+1$ und ist nun bereit Nachrichten dieser Sicht zu empfangen.

Backups die eine new-view Nachricht erhalten akzeptieren diese, wenn sie die richtige Signatur hat, das new-view Zertifikat korrekt ist und die Mengen $\mathcal{N}$ und $\mathcal{O}$ richtig berechnet sind. Wenn der backup Knoten die neue Sicht akzeptiert, sendet er für jede Nachricht in $\mathcal{N} \cup \mathcal{O}$ eine prepare Nachricht an das System und wechselt in die Sicht $v+1$.

2.5 Fairness

Um deadlocks zu vermeiden müssen Replikate die einen Request nicht ausführen können in die nächste Sicht wechseln. Allerdings muss eine Wartezeit eingehalten werden damit auch langwierige Anfragen eine Chance auf Durchführung haben. Damit nicht ein bloßes Wechseln der Sichten das System lahmlegen kann muss die Wartezeit auf eine Durchführung mit jedem Wechsel der Sicht ohne eine Abarbeitung einer Anfrage exponetiell vergrößert werden.

3 Erweiterungen

Der oben vorgestellte BFT-PK bietet Spielraum für etliche Optimierungen und Erweiterungen. Der Performance Falschenhals stellt sicher die Signierung der Nachrichten dar. Im Folgenden soll die Signatur durch eine Verschlüsselung per message authentication codes $MACs$ eines Knotens eine Performance Steigerung ermöglichen.

Die Beschränkung der Anzahl der Fehlerhaften Knoten auf f ist für langlaufende Systeme sehr schwer einzuhalten. Daher wurde ein Mechanismus implementiert der es ausgefallenen Clients ermöglicht sich selbst in einen stabilen Zustand zurückzusetzten. Dies wird proactive recovery genannt.

3.1 MACs statt public keys

Die Umstellung von signierten Nachrichten auf Verschlüsselung per MAC erscheint im ersten Moment trivial, jedoch erfordert diese Optimierung einige Veränderungen am Alogrithmus . Der Grund dafür ist, dass MACs weniger mächtig sind als die public key Methode (siehe erster Vortrag).

Daraus ergeben sich eine Reihe von Problemen, angefangen bei der Authentifizierung von Requests, pre-prepare, commit,u.ä. bis hin zu der Frage wie nun Zertifikate z.B. beim view-change übergeben werden.

Nachrichten werden in diesem Fall mit einem Vektor verschickt, der alle MACs für die jeweiligen Empfänger beinhaltet. Ein Knoten verifiziert die Richtigkeit der Nachricht, in dem er seinen MAC-Schlüssel mit dem aus dem Vektor vergleicht.

Die Request Authentifizierung wurde in BFT explizit implementiert. Die implizite Methode, wie sie im BFT-PK benutzt wurde funktioniert nun nicht mehr da Request von einigen Knoten identifiziert werden können und von anderen nicht. Daher weist der primary nur Requests Sequenznummern zu, deren Herkunft er verifizieren konnte. Backups ihrerseits akzeptieren nur pre-prepare Nachrichten die sie authentifizieren konnten. Das geschieht unter folgenden Bedingungen:

1. der MAC für Knoten i im Authentifizierungsvektor ist korrekt oder
2. i hat f prepare Nachrichten mit dem digest der Anfrage akzeptiert oder
3. i hat eine Anfrage von client c erhalten mit der gleichen Operation und dem gleichen Zeitstempel mit dem korrekten MAC im Authentifizierungsvektor.

Diese drei Bedingungen verhindern das Fälschen eines Requests durch einen fehlerhaften primary Knoten.

Die garbage collection funktioniert bei BFT ähnlich wie bei BFT-PK, jedoch ist ein schwaches Zertifikat bei der Verwendung von MACs nicht mehr ausreichend. Daher versucht jeder Knoten ein Quorum Zertifikat zu erhalten, um einen stable checkpoint zu erreichen.

View changes müssen nun anders ablaufen, da wie schon erwähnt Zertifikate nicht mehr übergeben werden können. Als Lösung versucht man hier schwache Zertifikate durch das Schicken von acknowledge Nachrichten auf eine view-change Nachricht zu simulieren.

3.2 Proactive Recovery

Um die Eigenschaften des Algorithmus zu erhalten und eine automatische Wiederherstellung der clients zu ermöglichen, gilt die Beschränkung für Fehlerhafte Knoten f im Folgenden für ein *kleines* Zeitfenster.

Um das zu ermöglichen werden folgende Annahmen gemacht:

- Sichere Cryptographie, d.h. einem Angreifer soll es nicht möglich sein Schlüssel zu erraten
- Read-Only Memory: Die Replikas speichern Schlüsselpaare anderer Knoten in einem nicht veränderbarem Speicher
- Watchdog Timer: jedes Replikat besitzt einen Timer der den laufenden Prozeß periodisch unterbricht und die Kontrolle an einen sog. recovery monitor übergibt. Diese entscheidet, ob ein Replikat fehlerhaft ist und rebootet ggf. das Repklikat.

Um nun ein Replikat wieder korrekt in das System einbinden zu können, überprüft der recovery monitor das log auf fehlerhafte Einträge und sendet recovery-requests an die übrigen Replikas , um sein log auf den aktuellen Stand zu bringen.

Nach einem Recovery soll der Knoten wieder voll einsatzfähig sein.

4 Zusammenfassung

Der Practical Byzantine Fault Tolerance Algorithmus von Miguel Castro ermöglicht faire und fehlertolerante redundante Ausführung von Service die sich durch endliche, deterministische Automaten darstellen lassen. Die Anzahl der Fehlerhaften Knoten muss bei höchstens $\frac{n-1}{3}$ liegen. Die Erweiterten Versionen bieten effiziente Implementierungen mit MACs und proactive Recovery, das unter anderem auch Schutz vor denial-of-service attacken gegen andere Knoten bietet.

Literatur

[Castro01] Miguel Castro. Practical Byzantine Fault Tolerance. Massachusetts, MIT Press, 2001.

[Castro00] M.Castro und B.Lyskov. Practical Byzantine Fault Tolerance. In Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI).San Diego,o.V., 2000