Self - Testing/Correcting Protocols

Carsten Rebbien
225 868

Stefan Richter

Zusammenfassung

In diesem Artikel werden selbsttestende sowie selbstkorrigierende Protokolle vorgestellt. Der Begriff Protokoll wird hier jedoch anders verwendet als es zum Beispiel von der Datenkommunikation her bekannt ist. Ei Protokoll sei hier die Zusammenfassung mehrerer Programme, die in einem Netzwerk von mehreren Prozessoren eine Funktion berechnen. Es wird ein “Selftester“ und ein “Selfcorrector“ für ein Protokoll, dass die sogenannte Byzantinische Agreement Funktion implementiert, beschrieben.

Inhaltsverzeichnis

1 Einführung

Das Testen von Programmen besteht normalerweise darin, das Programm mit verschiedenen Eingaben laufen zu lassen und zu überprüfen, ob die so erzielten Ergebnisse richtig sind oder nicht. Dazu müssen für diese Eingaben die richtigen Ergebnisse natürlich vorher bekannt sein. Ein Problem, welches sich einem hierbei stellt, ist die Art und Weise, wie man an die vermeintlich richtigen Ergebnisse kommt. Man könnte für einige wenige Eingaben „von Hand“ prüfen, ob das Ergebnis richtig ist. Jedoch kann man dann keine Aussage über die generelle Korrektheit des Programms machen. Eine andere Möglichkeit wäre die richtigen Ergebnisse mittels eines anderen Programms zu berechnen. Dieses muss dann wiederum ebenfalls auf Korrektheit überprüft werden.

Um dieses Problem zu umgehen, wurden sogenannte Programmchecker entwickelt. Ein Programmchecker ist ebenfalls ein Programm welches parallel zu dem eigentlichen Programm läuft und dessen Arbeit überprüft. Der Programmchecker prüft jedoch nur, ob das Programm auf der einen Eingabe, mit der es gestartet wurde, richtig läuft. Bei jedem Programmaufruf prüft der Programmchecker also erneut das Programm auf die jeweilige Eingabe hin. Ein selbsttestender Programmchecker (Selftester) ist ein Programm, welches des zu testende Programm P mit einer beliebigen Eingabe x aufruft und dann eine Wahrscheinlichkeit bestimmt, mit der $f(x) = P(x)$ gilt, wobei f die von P zu berechnende Funktion ist. Da ein Programmchecker bei gefundenem Fehlverhalten des zu prüfenden Programms selbst keine korrekte Lösung errechnet, werden sogenannte selbstkorrigierende Programme (Selfcorrector) verwendet. Ein solcher Selfcorrector korrigiert das fehlerhafte Programm, falls die Wahrscheinlichkeit für ein falsches Ergebnis (vom Selftester errechnet) vernachlässigbar klein ist.

Im folgenden werden Selftester bzw. -corrector für verteilte Protokolle vorgestellt, wobei der Begriff Protokoll sich hier auf eine Menge von n Programmen bezieht, die in einem Netzwerk von n Prozessoren eine verteilte Funktion f berechnen.

2 Begriffsklärung

Zunächst müssen noch die im weiteren Verlauf benutzten Begriffe erklärt sowie das zu Grunde liegende Modell beschrieben werden.

2.1 Begriffe/Definitionen

Verteilte Funktionen und Protokolle. Eine verteilte Funktion oder auch Problem Spezifikation sei hier $\Phi : V^n \rightarrow R^n$, wobei V ein beliebiger Definitionsbereich und R ein Wertebereich ist, für die gilt $V = R$. Die Berechnung dieser Funktion findet in einem vollständig verbundenen Netzwerk von n Prozessoren statt. Die Implementierung der n-stelligen Funktion wird Protokoll genannt. Sie ist nichts weiter als die Zusammenfassung der einzelnen Programme, die auf den einzelnen Prozessoren laufen. Da in einer solchen verteilten Umgebung davon auszugehen ist, dass Fehler auftreten können, sei es bedingt durch die Kommunikation zwischen den Prozessoren oder durch Fehlverhalten der Prozessoren selbst, muss dies in der Funktionsdefinition berücksichtigt werden. Das Ergebnis der Funktion sollte also nur von den fehlerfreien Prozessoren berechnet werden. Hierzu wird eine verteilte Funktion eingeführt, welche direkten Bezug auf ihre Umwelt, also auf eventuell fehlerhafte Prozessoren, nimmt: $f : W^n \rightarrow S^n$. f und Φ berechnen immer noch die gleiche Funktion. W enthält V und wird um Charakteristika erweitert, welche jeden einzelnen Prozessor etikettieren und direkten Einfluss auf das Ergebnis der Funktion nehmen. Genauso ist S die Erweiterung von R. E sei die Menge von Umgebungen in denen die Protokolle laufen. In einer Umgebung E werden nur bestimmte Ereignisse, die zum Fehlverhalten eines Prozessors führen können, zugelassen. Z. B. können Umgebungen definiert werden, in denen Prozessoren abstürzen (ausfallen) oder Nachrichten verspätet bzw. gar nicht verschickt werden können. Im weiteren Verlauf wird eine Umgebung betrachtet, in der Prozessoren ausfallen können.

Orakel Protokoll. Ein Orakel Protokoll ist ein Protokoll Π, welches ein anderes Protokoll P aufruft, das wiederum erst zur Laufzeit bekannt ist. Dabei kennt das Orakel Protokoll nicht die inneren Abläufe von P, sondern nur dessen Ergebnis. Π^P sei das Protokoll Π, welches P aufruft.

Sei weiterhin D_v die Wahrscheinlichkeitsverteilung, nach welcher die Eingaben v_i, $1 \leq i \leq n$, aus V bestimmt werden. $P[i]$ sei das Ergebnis, dass von Prozessor i in Protokoll P geliefert wird. Die Funktion $error(f, P, D_v, E)$ gibt die Wahrscheinlichkeit zurück, mit der $f \neq P$ ist, d.h. $f[i] \neq P[i]$ für mindestens ein i. Beta sei ein Vertrauensparameter.

Selbsttestendes Protokoll (Selftester). Sei $0 \leq \epsilon_1 < \epsilon_2 \leq 1$. Ein (ϵ_1, ϵ_2)-selbsttestendes Protokoll für f in Bezug auf D_v in E sei dann ein probalistisches Orakel Protokoll T_f, so dass für jedes P , dass in E läuft, mit β gilt:

1. $If\ error(f, P, D_v, \mathrm{E}) \leq \epsilon_1\ then\ Pr(\forall i : T_f^P[i] = \text{„}PASS\text{“}) \geq 1 - \beta$
2. $If\ error(f, P, D_v, \mathrm{E}) \geq \epsilon_2\ then\ Pr(\forall i : T_f^P[i] = \text{„}FAIL\text{“}) \geq 1 - \beta$

Selbstkorrigierendes Protokoll (Selfcorecctor). Sei $0 \leq \epsilon < 1$. Ein ϵ-selbstkorrigierendes Protokoll für f in Bezug auf D_v in E sei dann ein probalistisches Orakel Protokoll C_f, so dass für jedes P , dass in E läuft, mit Beta gilt:

$$If\ error(f, P, D_v, \mathrm{E}) \leq \epsilon\, then\ Pr(\forall i : C_f^P[i] = f[i]) \geq 1 - \beta$$

Komplexität. Die Komplexität des Selftesters/-correctors muss natürlich geringer sein, als die des zu prüfenden Protokolls. Da hier ein synchrones Netzwerk betrachtet wird, in der die Berechnung in mehreren Runden erfolgt, wird gefordert, dass die Laufzeit (Anzahl der Runden) von T_f^P (sowie C_f^P) $o(R)$ beträgt, wobei R die minimale worst-case Laufzeit des Protokolls P ist, welches f berechnet.

2.2 Die Byzantinische Agreement Funktion

Die Protokolle, die hier vorgestellt werden sollen, sind ein Selftester bzw. Selfcorrector für ein Protokoll, welches die sogenannte Byzantinische Agreement Funktion (BA) implementiert. Diese Funktion prüft, ob in einem Netzwerk von mehreren Prozessoren Einstimmigkeit herrscht, obwohl einige Prozessoren fehlerhaft arbeiten. Einstimmig bedeutet hier, dass sich alle Prozessoren auf den Inhalt verschickter Nachrichten einigen. Die BA-Funktion arbeitet folgendermaßen: ein als Quelle markierter Prozessor s verschickt einen Startwert v_s an alle Prozessoren im Netzwerk. Alle fehlerfreien Prozessoren müssen sich jetzt auf einen Wert einigen. Die ist, falls der Quellprozessor fehlerfrei war, der Startwert oder aber ein Defaultwert, falls der Quellprozessor nicht fehlerfrei war. Die Funktion stellt sich wie folgt dar: $BA : (V \times B)^n \rightarrow (V \times T)^n$, wobei V die Eingabemenge, $B = faulty, nonfaulty$ die möglichen Prozessorverhaltensweisen und $T = true, false$ die möglichen Terminierungszustände sind; und erfüllt folgende Bedingungen:

1. $\forall i\ b_i = nonfaulty$; *alle fehlerfreien Prozessoren einigen sich auf den gleichen Wert* und
2. $(d_i, t_i) = \begin{cases} (v_s, true)\ if\ b_s = nonfaulty \\ (v, true)\ where\ v \in V, otherwise \end{cases}$; *falls s fehlerfrei war, einigen sich alle fehlerfreien Prozessoren auf den Startwert.*

Einstimmigkeit kann jedoch nur erzielt werden, falls die Anzahl der fehlerhaften Prozessoren nicht höher als ein Drittel der Gesamtzahl der Prozessoren ist. Es ist möglich, dass ein Prozessor, vorrausgesetzt er kennt den Schwellwert für fehlerhafte Prozessoren, herausfindet, dass der sendende Prozessor fehlerhaft ist. Dies kann erreicht werden, wenn es zu viele widersprüchliche Versionen des gesende-

ten Startwerts gibt. Der betreffende Prozessor weiss dann ausdrücklich, dass der Quellprozessor fehlerhaft war.

In dem Selftester für die BA-Funktion wird auch die sogenannte Crusader Funktion (CA) benutzt. Diese Funktion arbeitet genau wie die BA-Funktion, jedoch unterscheidet sie sich in der zweiten Bedingung. Diese lautet hier: *alle fehlerfreien Prozessoren, die nicht ausdrücklich wissen, dass s fehlerhaft ist, einigen sich auf einen Wert*. Man sieht also, dass wenn s fehlerhaft ist, die CA-Funktion alle Prozessoren in drei Gruppen aufteilt: die Prozessoren, die fehlerhaft sind; die, die fehlerfrei sind und ausdrücklich wissen, dass s fehlerhaft ist; und die restlichen fehlerfreien Prozessoren, die sich auf einen Wert einigen.

3 Self-Testing/Correcting Protocols - Beispiele

Im folgenden wird angenommen, dass die Aufrufe des zu testenden bzw. korrigierenden Protokolls sequentiell geschehen. D.h. beginnend mit einem initialen Zustand des Systems ruft der Selftester bzw. Selfcorrector das Protokoll mehrmals hintereinander auf, wobei angenommen wird, dass jeder Prozessor einen nicht flüchtigen Speicher besitzt, welcher Informationen über mehrere Aufrufe hinweg speichert. So können Schlüsse bezüglich vorhergehender Aufrufe zugelassen werden. Nach jedem Aufruf des Protokolls werden fehlerhafte Prozessoren neu gebootet. P sei das zu testende bzw. korrigierende Protokoll, welches die BA-Funktion implementiert.

3.1 Selftester für die BA-Funktion

Zunächst wird ein einfacher Selftester (Abbildung 1) für einen Durchlauf von P betrachtet. Dieser einfache Selftester dient dann als Basis für den eigentlichen Selftester.

Abbildung 1 zeigt den Code des einfachen Selftesters für einen Prozessor i. Als erstes sucht der Quellprozessor sich einen Startwert gemäß der Verteilung D_v aus. Dieser wird daraufhin mittels des ersten CA-Aufrufs an alle Prozessoren verschickt. Wenn s fehlerfrei ist, kennen jetzt alle fehlerfreien Prozessoren im Netzwerk diesen Startwert, ist s jedoch fehlerhaft, so kennt jeder Prozessor entweder den seinen Input oder die Tatsache, dass s fehlerhaft ist. In diesem Fall müsste in *input* der Defaultwert stehen. Als Defaultwert wird hier $\perp$ angenommen. Als nächstes wird das P aufgerufen, woraufhin jeder Prozessor seinen erhaltenen Wert mittels der CA-Funktion wiederum an alle Prozessoren schickt. In G werden dann

alle fehlerfreien Prozessoren gespeichert und überprüft, ob der Quellprozessor tatsächlich fehlerhaft war oder nicht. Jeder Prozessor entscheidet sich dann anhand seiner Sicht derInput- und Outputwerte für entweder *"fail"* oder *"not fail"*.

Protocol Simple_Agreement_Tester(P, s)

s chooses $v \in_{D_V} V$
$input \leftarrow CA(s, v)$
$P(s, v)$
for $i = 1, \cdots, n$
    $output_i \leftarrow CA(i, P[i])$
let $G = \{i$: message received in $CA(i, P[i])\}$
if message received in $CA(s, P[s])$
    $sender\_alive \leftarrow$ **true**
$fail \leftarrow$ $(\exists\, j, k \in G : output_j \neq output_k) \vee$
    $(\exists\, j \in G : (output_j \neq input \wedge sender\_alive) \vee$
        $(output_j \neq input \wedge output_j \neq \perp))$

Abbildung 1: einfacher Selftester für die Byzantinische Agreement Funktion für einen Durchlauf; für Prozessor i

Dieser einfache Selftester wird nun als Basis für einen allgemeinen Selftester benutzt. Wie in Abbildung 2 (auch hier wird nur der Code für einen Prozessor dargestellt) zu sehen ist, wird der einfache Selftester nur angemessen oft aufgerufen, anhand von ähnlichen Ergebnissen eine Schlussfolgerung zu treffen.

Sei nun $Y_l[i], 1 \leq l \leq N$, eine Zufallsvariable mit Werten 0 oder 1, welche anzeigt, ob für einen Prozessor *"fail"* ausgegeben wurde oder nicht. Sei weiterhin $\mu_i = E[Y_l[i]] = \delta \cdot error(BA, P, D_v, C)$ der Erwartungswert dieser Zufallsvariablen, wobei $\delta > 1/2$. Die folgenden Korrolare sind notwendig, um das darauf folgende Theorem zu beweisen.

Korrolar 1. *Sei* $\mu' \leq \mu_i$, *fr alle* i, *und sei* $N = \frac{1}{\mu'} \cdot 16ln(\frac{2n}{\beta})$. *Dann* $Pr(\exists i : Y[i] \leq \frac{\mu'}{2}) \leq \beta$, *wobei* $Y[i] = \sum_{l=1}^{N} \frac{Y_l[i]}{N}$.

Korrolar 2. *Sei* $\mu'' \geq \mu_i$, *fr alle* i, *und sei* $N = \frac{1}{\mu''} \cdot 4ln(\frac{2n}{\beta})$. *Dann* $Pr(\exists i : Y[i] \geq 2\mu'') \leq \beta$, *wobei* $Y[i] = \sum_{l=1}^{N} \frac{Y_l[i]}{N}$.

Theorem 1. *Das Protokoll aus Abbildung 2 ist ein* $(\frac{\epsilon}{8}, \epsilon)$*-Selftester für die BA-Funktion.*

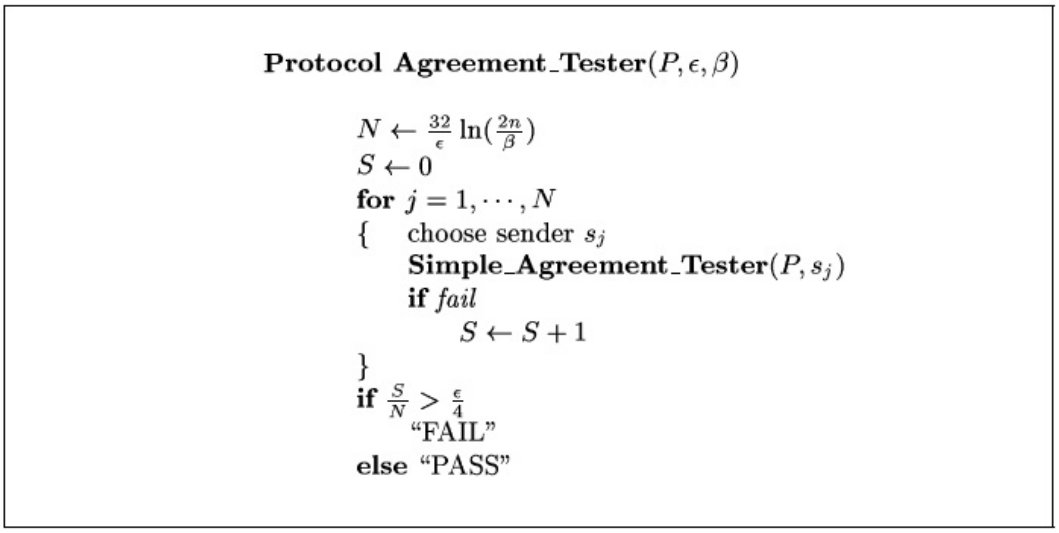

Abbildung 2: Selftester für die Byzantinische Agreement Funktion; für Prozessor i

Beweis (Skizze). *Da die CA-Funktion nur O(1) Runden benötigt, ist die Komplexitätsbedingung erfüllt. Mit einen erwarteten Ergebnis von $E[Y[i]] = \mu_i = \delta \cdot error$ an Prozessor i werden die folgenden zwei Fälle betrachtet:*

error $\geq \epsilon$: *Sei $\mu' = \frac{\epsilon}{2}$ (die untere Schranke von δ) und $N = \frac{1}{\epsilon} \cdot 32ln(\frac{2n}{\beta})$. Nach Korrolar 1 ist $Pr(\exists i : Y[i] \leq \frac{\epsilon}{4}) \leq \beta$. Das Protokoll aus Abbildung 2 gibt jedoch "FAIL" für Prozessor i aus, falls $Y[i] > \frac{\epsilon}{4}$. Also gibt das Protokoll mit einer Wahrscheinlichkeit von mindestens $1 - \beta$ "FAIL" für jeden Prozessor aus, falls error $\geq \epsilon$.*

error $\leq \frac{\epsilon}{8}$: *Sei $\mu'' = \frac{\epsilon}{8}$ (gleichbedeutend mit $\delta = 1$) und $N = \frac{1}{\epsilon} \cdot 32ln(\frac{2n}{\beta})$. Nach Korrolar 2 ist $Pr(\exists i : Y[i] \geq \frac{\epsilon}{4}) \leq \beta$. Das Protokoll aus Abbildung 2 gibt jedoch "PASS" für Prozessor i aus, falls $Y[i] < \frac{\epsilon}{4}$. Also gibt das Protokoll mit einer Wahrscheinlichkeit von mindestens $1 - \beta$ "PASS" für jeden Prozessor aus, falls error $\leq \frac{\epsilon}{8}$.*

3.2 Selfcorrector für die BA-Funktion

Falls die vom Selftester geschätzte Fehlerwahrscheinlichkeit klein genug ist, kann der folgende Selfcorrector dazu benutzt werden, das richtige Ergebnis von P für einen Eingabewert x zu bestimmen. Angenommen diese Fehlerwahrscheinlich-

keit sei kleiner als als 0,05 (also: error$(BA, P, D_v, E) \leq \epsilon = 0,05$). Der Code dieses Selfcorrectors für einen Prozessor i ist in Abbildung 3 zu sehen. Die Funktionsweise des Selfcorrectors wird im Wesentlichen durch die zwei Aufrufe von P bestimmt. Der erste Aufruf von P erfolgt mit einem zufällig gewählten Startwert. Der zweite mit der Differenz aus dem Eingabewert und diesem zufällig gewählten Startwert. Falls beide Aufrufe von P korrekt durchlaufen, d.h. alle Prozessoren einigen sich auf den Startwert, wird die Summe der beiden Ergebnisse gebildet, ansonsten der Defaultwert angenommen. Dieser Vorgang wird angemessen oft wiederholt und das am meisten vorkommende Ergebnis als Output für Prozessor i ausgegeben.

Protocol Agreement_Corrector$(P, s, x, \epsilon, \beta)$

$N \leftarrow O(\frac{\ln(n/\beta)}{\epsilon})$
for $j = 1, \cdots, N$
{ s chooses $r \in_{D_V} V$
$output_1 \leftarrow P(r)$
$output_2 \leftarrow P(x -_V r)$
if $(output_1 = \perp \wedge output_2 = \perp)$
$outcome_j \leftarrow \perp$
elseif $(output_1 \neq \perp \wedge output_2 \neq \perp)$
$outcome_j \leftarrow output_1 +_V output_2$
else $outcome_j \leftarrow$ '$*$'
}
$P[i] \leftarrow$ most common $outcome_j$, $1 \leq j \leq N$,
s.t. $outcome_j \neq$ '$*$'

Abbildung 3: Selfcorrector für die Byzantinische Agreement Funktion; für Prozessor i

Mit der nachfolgenden Proposition lässt sich das darauf folgende Theorem beweisen. Der Beweis für diese Proposition folgt direkt aus den Chernoff-Schranken, was hier aber nicht weiter relevant sein soll.

Proposition. *Sei $\alpha \geq \frac{3}{4}$, und seien $X_1, X_2, .., X_N$ gleichverteilte binäre Zufallsvariablen, so dass $Pr[X_i = 1] \geq \alpha, i = 1, 2, ..., N$. Dann*

$$Pr\left[\sum_{i=1}^{N} X_i > \frac{2N}{3}\right] \geq 1 - e^{-\frac{\alpha N}{324}}.$$

Theorem 2. *Das Protokoll aus Abbildung 3 ist ein ϵ-Selfcorrector für die BA-Funktion.*

Beweis (Skizze). *Angenommen die Wahrscheinlichkeit, dass bei einem Durchlauf ein Aufruf von P kein korrektes Ergebnis liefern (also $P[i] \neq r$ oder $P[i] \neq x-r$), sei höchstens ϵ. Dann liefern beide Aufrufe von P mit einer Wahrscheinlichkeit von mindestens $1-2\epsilon$ ein korrektes Ergebnis zurück. Sei nun $\alpha = 1-2\epsilon$. Mit einer ausreichend hohen Anzahl von Durchläufen, so dass dies für alle Prozessoren eintritt, folgt das Theorem direkt aus Proposition 1.*

4 Zusammenfassung

Nach selbsttestenden bzw. selbstkorrigierenden Programmen für meist numerische Probleme ist dies der erste Ansatz für selbsttestende bzw. selbstkorrigierende Protokolle, also verteilte Programme. Diese Selftester bzw. Selfcorrector rufen das zu testende bzw. korrigierende Protokoll selbständig in mehreren Runden auf und arbeiten dann mit den erhaltenen Ergebnissen weiter. Dies wurde durch die einfachen Beispiele, den Selftester sowie den Srlfcorrector für die Byzantinische Agreement Funktion, deutlich gemacht.

Literatur

[1] BLUM, MANUEL, MICHAEL LUBY und RONITT RUBINFELD: *Self-Testing/Correcting with Applications to Numerical Problems*. In: *STOC'90*, 1990.

[2] DOLEV, DANNY: *The Byzantine Generals Strike Again*. In: *Journal of Algorithms, 3(1):14-30*, 1982.

[3] FRANKLIN, MATTHEW, JUAN A. GARAY und MOTI YUNG: *Self-Testing/Correcting Protocols*. In: *LNCS 1963*, 1999.