

Rheinisch-Westfälische Technische Hochschule Aachen
Lehrstuhl für Informatik II

Proseminar Verteilte Algorithmen, WS 2006/2007

Das „Cheating Husbands“-Puzzle

eine Einführung

von *David R. Piegdon*

19. Januar 2007

Betreuer: Daniel Willems



Diese Ausarbeitung stützt sich vor allem auf [MDH85].

Der Autor, Student der Informatik an der *Rheinisch-Westfälische Technische Hochschule Aachen*, ist erreichbar via Email. Außerdem kann die Ausarbeitung nach der Korrektur auf seiner Homepage gefunden werden.

David Rasmus Piegdon
david.rasmus.piegdon@rwth-aachen.de
<http://david.piegdon.de/products.html>

Dieser Text wurde unter *Linux* mit *vim* geschrieben, und mit \LaTeX gesetzt.

Inhaltsverzeichnis

1 Einführung	3
1.1 Verteilte Systeme	3
1.2 Wissen, Kommunikation und Aktion	4
2 Synchronisation in verteilten Systemen	4
2.1 Asynchrone Kommunikation	6
2.2 Schwach- <i>b</i> -synchrone Kommunikation	6
2.3 Stark- <i>b</i> -synchrone Kommunikation bei Systemen mit Ausfällen	10
3 Quick Elimination - eine minimale Lösung	11
4 Ergebnisse	14
Abbildungsverzeichnis	14
Literatur	14

1 Einführung

Der ursprüngliche Text [MDH85], mit dem sich diese Ausarbeitung hauptsächlich beschäftigt, analysiert als Ergänzung zu [HM84] exemplarisch Auswirkungen von und Zusammenhänge zwischen Wissen, Kommunikation und Aktion in verschiedenartigen verteilten Systemen. Diese Systeme haben dabei gemein, dass alle ihre Komponenten (mindestens) ein Bit an Speicher teilen.

Es wird ein Algorithmus für Systeme mit synchroner, verlustloser Kommunikation zur Synchronisation einer Integer-Variable entwickelt. Der Wert der Variable darf zwischen zwei Komponenten um maximal 1 abweichen. Desweiteren wird untersucht, welche Auswirkungen asynchrone und schwach synchrone Kommunikationssysteme oder fehlerhafte verteilte Systeme auf den Algorithmus haben. Zuletzt wird ein Minimalzeit-Algorithmus für synchrone, verlustlose Systeme vorgestellt.

1.1 Verteilte Systeme

Als verteiltes System werden Systeme bezeichnet, die aus mehreren Prozessen, Prozessoren oder Computern bestehen. Dabei wird jede dieser Einheiten als *Komponente* bezeichnet. Jede dieser Komponenten muss nebenläufig sein, also unabhängig von den anderen Komponenten ihre Aufgaben erfüllen.¹ Außerdem hat im Allgemeinen jede Komponente unvollständiges Wissen über den Gesamtzustand des Systems und es fehlt möglicherweise eine zeitliche Synchronisation der Komponenten. Dadurch kann das Gesamtsystem nichtdeterministisch sein, da bei Verwendung nebenläufiger Prozesse für ein Problem unterschiedliche zeitliche Abfolgen mit unterschiedlichen Zwischen- oder Endergebnissen möglich sind und diese Abfolgen nicht unbedingt hervorsagbar sind.

¹Diese kurze Definition stammt aus [Tel00]. Es gibt weitere, verschiedene Definitionen, abhängig von der Betrachtungsweise und der Notwendigkeit, Algorithmen aus dem Feld der verteilten Systeme zu nutzen.

Verteilte Systeme werden für verschiedenste Zwecke genutzt, daher ist es zwingend nötig, sich mit den Eigenschaften und Fähigkeiten solcher Systeme auseinanderzusetzen. Zu ihren Vorteilen gehört, dass sie meist sehr gut skalierbar und fehlertoleranter als zentrale Systeme sind. Wenn zum Beispiel eine Komponente ausfällt, kann eine andere ihre Arbeit übernehmen oder die fehlerhafte Komponente durch eine neue ersetzt werden, ohne dass das System zusammenbricht; je nach Design des Systems können auch Teile des Gesamtsystems ohne diese Komponente weiter arbeiten. Die weiter oben beschriebenen Eigenschaften machen es nötig, komplett neue Modelle zu entwickeln, da man mit klassischen Modellen Verhalten und Funktionsweise verteilter Systeme nur unzureichend analysieren kann.

1.2 Wissen, Kommunikation und Aktion

In deterministischen Systemen wie Prozessoren - oder abstrakter: einzelnen Komponenten in verteilten Systemen - ist jede Handlung auf das Wissen gestützt, das diese Komponente zu dem Zeitpunkt der Handlung hat. Kommunikation dient dazu, dieses Wissen zwischen einzelnen Komponenten auszutauschen. Das „*Cheating Husbands*“-Puzzle ist als „Grundlagenforschung“ im Bereich verteilte Systeme zu betrachten: Was wird möglich, wenn alle Komponenten ein Bit an Speicher teilen? Wie wirken sich fehlende oder schwache Synchronisation oder ausgefallene Komponenten aus? Was müssen die Komponenten an Wissen mitbringen, damit sie sich synchronisieren können? Dabei spielt, wie sich zeigen wird, der Begriff des Allgemeinwissens eine zentrale Bedeutung.

2 Synchronisation in verteilten Systemen

Das folgende Rätsel stellt ein Synchronisationsproblem verteilter Systeme dar. Durch Variation wird untersucht, welche Auswirkungen asynchrone, schwach synchrone und fehlerbehaftete verteilte Systeme auf die Lösung haben.

Im matriarchalischen Stadtstaat Mamajorca herrscht ein Ehebruch-Problem. Um dem Einhalt zu gebieten, entschließt sich die Königin Henrietta I, alle Ehefrauen des Stadtstaates auf dem Marktplatz zu versammeln und richtet dort die folgenden Worte an sie:

- (1) Mindestens einer eurer Ehemänner betrügt seine Frau.
- (2) Keine von euch weiß von ihrem eigenen Mann, ob er seine Frau betrügt, aber jede von euch weiß von allen anderen Männern, die ihre Frau betrügen.
- (3) Ihr dürft nicht mit Anderen über die Treue oder Untreue von euren Männern reden.
- (4) Wenn ihr herausfindet, dass euer Mann euch betrügt, so müsst ihr ihn am gleichen Tag um Mitternacht erschießen.

Alle verheirateten Frauen in Mamajorca sind der perfekten Logik fähig und ihrer Königin hörig; dies ist auch Allgemeinwissen. Durch die Versammlung aller Frauen werden die Worte der Königin ebenfalls zu Allgemeinwissen (alle wissen, was die Königin gesagt hat; alle wissen, dass alle wissen, was die Königin gesagt hat; alle wissen, dass alle wissen, dass alle...).

Als n Nächte später mitternachts Schüsse gehört werden, sind viele der Frauen erleichtert. Auch das Problem der betrügerischen Ehemänner wurde zunächst aus der Welt geschafft.

Was hat die Königin ihren Untertanen wichtiges gesagt, dass diese fähig waren, treue von untreuen Männern zu unterscheiden? Wieviele Männer wurden erschossen? Betrachten wir das, was sie ihren Untertanen gesagt hat und was diese schon wissen, etwas genauer: Die Frauen wissen schon vorher, welche der *anderen* Männer untreu sind, jedoch erst die Kundgabe der Königin macht dies zu Allgemeinwissen (d.h. alle Frauen wissen, dass alle Frauen wissen, dass alle . . . , welche Männer untreu sind, ausgenommen den Eigenen).

Gibt es nach dieser Regel nur einen untreuen Mann, so weiß seine Frau von keinem untreuen Mann, alle anderen Frauen wissen von einem Mann. Gibt es m untreue Männer, so wissen m Frauen von $m - 1$ untreuen Männern und die restlichen Frauen wissen von m untreuen Männern. Es ist also bei allen Frauen der genaue Wert von m unbekannt.

Betrachten wir das Problem etwas mathematischer, so kann man es als Synchronisationsproblem verschiedener Komponenten eines verteilten Systems abstrahieren, bei dem eine Integer-Variable (m) synchronisiert werden soll, die sich zwischen zwei beliebigen Komponenten um maximal 1 unterscheidet und alle Komponenten ein Bit an Speicher teilen. $\lceil m \rceil$ sei der größere bekannte Wert von m , $\lfloor m \rfloor$ sei der kleinere. $\lceil m \rceil$ -Komponenten seien die Komponenten, die den größeren Wert kennen, $\lfloor m \rfloor$ -Komponenten diejenigen, die den kleineren kennen.

Die Mitteilung der Königin an ihre Untertanen entspricht dann folgender Mitteilung an alle Komponenten des Systems:

- (1) Mindestens eine Komponente kennt einen niedrigeren Wert für m als eine andere Komponente.
- (2) Jede Komponente weiß, wieviele Komponenten den minimalen Wert ($\lfloor m \rfloor$) kennen, sich selber nicht mitgezählt.
- (3) Kommunikation über den Wert der Variable oder die Anzahl der $\lceil m \rceil$ -Komponenten oder $\lfloor m \rfloor$ -Komponenten ist nicht zugelassen.
- (4) Wenn nachvollziehbar wird, dass es eine Komponente gibt, die einen größeren Wert ($\lceil m \rceil$) kennt, muss der größere Wert angenommen werden und allen anderen Komponenten unmittelbar mitgeteilt werden, dass dies die letzte Synchronisationsrunde ist (indem das geteilte Speicherbit gesetzt wird).

Gibt es in einem verteilten System insgesamt k Komponenten und genau d $\lfloor m \rfloor$ -Komponenten, so wissen diese d Komponenten folglich von $(d - 1)$ $\lfloor m \rfloor$ -Komponenten. Die anderen $k - d$ ($\lceil m \rceil$ -Komponenten) hingegen wissen von d $\lfloor m \rfloor$ -Komponenten.

Wenn $m = 1$, kann eine einzelne Komponente unmittelbar nach (2) und (1) schließen, dass alle anderen Komponenten einen größeren Wert kennen, da die Komponente selber von keiner anderen Komponente weiß, die einen minimalen Wert kennt. Diese Komponente wird also den neuen Wert für m annehmen und allen anderen Komponenten das Ende der

Synchronisation über das geteilte Speicherbit signalisieren. Dieses Signal ist das Äquivalent zum von allen gehörten Schuss in der Geschichte aus Mamajorca.

Satz: Gibt es $d \geq 1$ $\lfloor m \rfloor$ -Komponenten in einem System, so können diese Komponenten in der d -ten Synchronisationsrunde feststellen, dass es eine Komponente gibt, die einen größeren Wert für m kennt als sie selber.

Beweis durch vollständige Induktion: (Induktionsstart) Der Algorithmus funktioniert für $d = 1$ (siehe oben). (Induktionsvoraussetzung) Der Algorithmus funktioniert für d . (Induktionsschritt) Für $d+1$ gilt: alle $\lfloor m \rfloor$ -Komponenten wissen von d $\lfloor m \rfloor$ -Komponenten. Da jedoch in der d -ten Synchronisationsrunde das geteilte Speicherbit nicht gesetzt wird, kann jede Komponente² schließen, dass die $\lfloor m \rfloor$ -Komponenten von d $\lfloor m \rfloor$ -Komponenten wissen und nicht von $d-1$. Jede $\lfloor m \rfloor$ -Komponente kann also schließen, dass sie selbst eine $\lfloor m \rfloor$ -Komponente ist; sie wird also in der $d+1$ -ten Synchronisationsrunde den neuen Wert $\lfloor m \rfloor$ für m annehmen und das geteilte Speicherbit setzen.

Wenden wir dieses Ergebnis auf die Geschichte aus Mamajorca an, können wir folgern, dass, wenn in der n -ten Nacht Schüsse fallen, $m = n$ untreue Männer erschossen wurden und dies *alle* untreuen Männer waren. Die eigentlich wichtigen Informationen, die die Königin ihren Untertanen gegeben hat, sind also:

- Es gibt mindestens einen betrügerischen Ehemann ($m \geq 1$).
- Es ist ab sofort Allgemeinwissen, dass alle Frauen von ihrem eigenen Mann nicht wissen, ob er untreu ist, es aber von allen anderen wissen.
- Jede Frau wird ihren Ehemann an dem Tag mitternachts erschießen, an dem sie herausfindet, dass er sie betrügt.
- Diese Regelung gilt ab sofort und für alle gleichzeitig. (Synchronisationsstart)

2.1 Asynchrone Kommunikation

Asynchrone Kommunikationssysteme sind hier Systeme, die Nachrichten in jedem Fall zustellen, jedoch für die Zustellungszeit keine obere Schranke haben. Folglich kommen Nachrichten *irgendwann* an, es kann jedoch keine Aussage darüber gemacht werden, ob sie zu einem *bestimmten* Zeitpunkt schon angekommen sind oder ankommen werden.

Es kann schnell gezeigt werden, dass der oben angegebene Synchronisationsalgorithmus bei Verwendung asynchroner Kommunikationssysteme *nicht* funktioniert:

Für den Fall $d = 2$ gibt es zwei $\lfloor m \rfloor$ -Komponenten, die beide wissen, dass die jeweils andere Komponente eine $\lfloor m \rfloor$ -Komponente ist. Da eine Komponente aber nicht wissen kann, wann die andere Komponente die Nachricht (d.h. den Synchronisationsstart) erhalten hat bzw. erhalten wird, kann sie nicht feststellen, wann die andere Komponente das geteilte Speicherbit setzen würde, gäbe es insgesamt nur eine $\lfloor m \rfloor$ -Komponente. Beide Komponenten müssen eigentlich auf die jeweils andere Komponente warten, da nicht ausgeschlossen werden kann, dass $d = 1$. Da sie aber nicht wissen, *wie lange* sie warten müssen, kann der Algorithmus hier nicht funktionieren.

2.2 Schwach- b -synchrone Kommunikation

Kommunikationssysteme, die garantieren, dass eine Nachricht spätestens $(b-1)$ Zeiteinheiten ankommt, nachdem sie abgeschickt wurde, werden hier schwach-synchrone Kom-

²insbesondere die $\lfloor m \rfloor$ -Komponenten

munikationssysteme mit der Grenze b oder *schwach- b -synchrone Kommunikationssysteme* genannt.

Satz (a): Gibt es $d \geq 1$ $\lfloor m \rfloor$ -Komponenten, so kann eine Komponente nach $b \cdot (d - 1)$ Synchronisationsrunden schließen, dass es d $\lfloor m \rfloor$ -Komponenten gibt, wenn ein schwach- b -synchrones Kommunikationssystem verwendet wird.

Mit anderen Worten: eine Komponente κ , die von d_κ $\lfloor m \rfloor$ -Komponenten weiß, kann nach $b \cdot d_\kappa$ Runden³ schließen, dass es $d = d_\kappa + 1$ $\lfloor m \rfloor$ -Komponenten gibt und sie selber eine solche Komponente ist.

Beweis: Eine Komponente, die von 0 $\lfloor m \rfloor$ -Komponenten weiß, kann unmittelbar nach dem Erhalt des Startsignals schließen, dass sie selber die einzige $\lfloor m \rfloor$ -Komponente ist (also $d = 1$). Sie wird umgehend das Synchronisationsende signalisieren und $\lfloor m \rfloor$ als neuen Wert übernehmen. Zu beachten ist hierbei, dass diese Komponente das Startsignal bis zu $(b - 1)$ Runden nach Sendung des Signals erhalten haben könnte, also auch bei nur einer einzigen $\lfloor m \rfloor$ -Komponente erst nach $(b - 1)$ Runden das Ende signalisiert worden sein könnte. Eine Komponente, die von *einer* $\lfloor m \rfloor$ -Komponente weiß, kann also erst in der b -ten Runde ohne Signal schließen, dass sie selber eine $\lfloor m \rfloor$ -Komponente ist, denn es ist möglich, dass sie das Startsignal unmittelbar nach Sendung erhalten hat (also zum frühest möglichen Zeitpunkt), die eine ihr bekannte $\lfloor m \rfloor$ -Komponente aber erst nach $(b - 1)$ Runden (also zum letz möglichen Zeitpunkt). Gilt nun die Behauptung für d , so kann eine Komponente, die d $\lfloor m \rfloor$ -Komponenten kennt, ähnlich erst nach $b(d - 1) + b = b \cdot d$ Runden schließen, dass es $(d + 1)$ $\lfloor m \rfloor$ -Komponenten gibt, sie selber also eine $\lfloor m \rfloor$ -Komponente ist. Nach vollständiger Induktion ist Behauptung (a) also korrekt.

Angenommen, es gibt genau zwei Komponenten α und β . Komponente α erhält das Startsignal in Runde n und weiß, dass Komponente β eine $\lfloor m \rfloor$ -Komponente ist. Wenn nun in Runde $(n + 1)$ von β das Ende signalisiert wird, könnte α dennoch eine $\lfloor m \rfloor$ -Komponente sein: es könnte sein, dass β das Startsignal in Runde $(n - 1)$ erhalten hat und $d_\beta \cdot b = 2$ Runden gewartet hat, weil β weiß, dass α eine $\lfloor m \rfloor$ -Komponente ist; es könnte aber auch sein, dass β das Signal in Runde $(n + 1)$ erhalten hat und, weil es von keiner $\lfloor m \rfloor$ -Komponente weiß, unmittelbar das Synchronisationsende signalisiert hat.

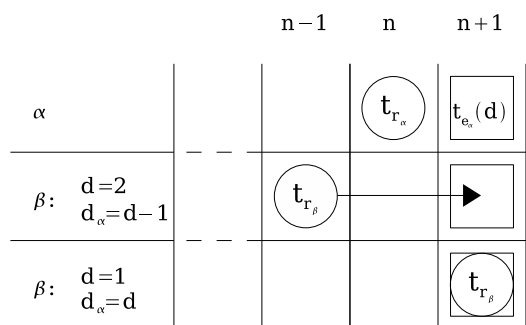


Abbildung 1: Zwei verschiedene Erklärungen für das Verhalten von β

Die Zeitintervalle, in denen eine Komponente als $\lfloor m \rfloor$ - oder $\lceil m \rceil$ -Komponente zugeordnet werden kann, überschneiden sich also für die Fälle $d = 1$ und $d = 2$. Man kann nun versuchen, diese Intervalle soweit zu manipulieren, dass sie disjunkt werden.

Betrachten wir dazu etwas allgemeiner Komponente α mit $d \geq 1$ beliebig und $b \geq 2$

³also in Runde $t_e = d_\kappa \cdot b + 1$

beliebig: die *früheste* Runde, in der eine $\lfloor m \rfloor$ -Komponente das Startsignal erhält, nennen wir im Folgenden die erste *signifikante* Runde t_s . Angenommen, α empfängt das Startsignal zum Zeitpunkt t_{r_α} , so muss es zwischen $t_{r_\alpha} - (b - 1)$ und t_{r_α} losgeschickt worden sein. Folglich haben alle Komponenten das Startsignal irgendwann in $[t_{r_\alpha} - (b - 1), t_{r_\alpha} + (b - 1)]$ erhalten, also ist $t_s \in [t_{r_\alpha} - (b - 1), t_{r_\alpha} + (b - 1)]$.

Jede Komponente κ kennt diese für sie relativen Grenzen. Außerdem weiß sie auch, dass die Anzahl der $\lfloor m \rfloor$ -Komponenten $d \in \{d_\kappa, d_\kappa + 1\}$ ist, abhängig davon, ob sie selber zu dieser Gruppe gehört oder nicht. (d_κ ist hierbei die Anzahl der Komponente κ bekannten $\lfloor m \rfloor$ -Komponenten.)

Stellen wir nun die relevanten Zeitintervalle auf, in denen für d_α und $d_\alpha + 1$ das Ende signalisiert wird. t_e sei die Runde, in der das Ende signalisiert wird, gezählt vom Versenden des Startsignals an (wie oben beschrieben gilt hier $t_e(d) = b(d - 1) + 1$). Betrachten wir die Intervalle außerdem aus der Sichtweise von α :

$$\begin{aligned} t_{e_\alpha}(d) &= t_s + t_e(d) \\ \Rightarrow t_{e_\alpha}(d) &\in \left[\underbrace{t_{r_\alpha} - (b - 1)}_{\inf(t_s)} + t_e(d), \underbrace{t_{r_\alpha} + (b - 1)}_{\sup(t_s)} + t_e(d) \right] \end{aligned}$$

Gibt es $d = d_\alpha$ $\lfloor m \rfloor$ -Komponenten, so wird eine Komponente in Runde $t_e(d_\alpha) = b(d_\alpha - 1) + 1$ das Ende signalisieren:

$$\begin{aligned} t_{e_\alpha}(d_\alpha) &\in \left[\underbrace{t_{r_\alpha} - b + 1}_{\inf(t_s)} + \underbrace{b(d_\alpha - 1) + 1}_{t_e(d_\alpha)}, \underbrace{t_{r_\alpha} + b - 1}_{\sup(t_s)} + \underbrace{b(d_\alpha - 1) + 1}_{t_e(d_\alpha)} \right] \\ &= [t_{r_\alpha} + b(d_\alpha - 2) + 2, t_{r_\alpha} + b d_\alpha] \end{aligned} \quad (1)$$

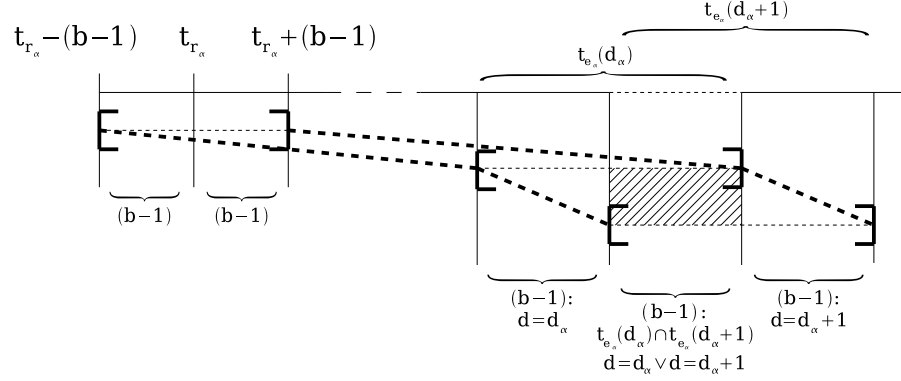
Gibt es aber $d = d_\alpha + 1$ $\lfloor m \rfloor$ -Komponenten, so gilt $t_e = b(d_\alpha + 1 - 1) + 1$:

$$\begin{aligned} t_{e_\alpha}(d_\alpha + 1) &\in \left[\underbrace{t_{r_\alpha} - b + 1}_{\inf(t_s)} + \underbrace{b(d_\alpha) + 1}_{t_e(d_\alpha)}, \underbrace{t_{r_\alpha} + b - 1}_{\sup(t_s)} + \underbrace{b(d_\alpha) + 1}_{t_e(d_\alpha)} \right] \\ &= [t_{r_\alpha} + b(d_\alpha - 1) + 2, t_{r_\alpha} + b(d_\alpha + 1)] \end{aligned} \quad (2)$$

Bildet man nun (1) \cap (2),

$$\begin{aligned} &\overbrace{[t_{r_\alpha} + b(d_\alpha - 2) + 2, t_{r_\alpha} + b d_\alpha]}^{(1)} \cap \overbrace{[t_{r_\alpha} + b(d_\alpha - 1) + 2, t_{r_\alpha} + b(d_\alpha + 1)]}^{(2)} \\ &= [t_{r_\alpha} + b(d_\alpha - 1) + 2, t_{r_\alpha} + b d_\alpha] \\ &= \underbrace{[t_{r_\alpha} + b(d_\alpha - 1) + 2]}_{:=h}, \underbrace{[t_{r_\alpha} + b(d_\alpha - 1) + 2 + b - 2]}_{=h} \end{aligned}$$

so erhält man ein $(b - 1)$ Runden umfassendes Intervall, wie in Abbildung 2 verdeutlicht. Wenn in diesem Intervall das Ende signalisiert wird, können alle Komponenten, die nicht selber das Ende signalisieren, nicht feststellen, zu welcher Gruppe sie gehören, denn es könnte sowohl $d = d_\alpha$ als auch $d = d_\alpha + 1$ sein.

Abbildung 2: Sich überschneidende Intervalle für d_α und $d_\alpha + 1$.

Es kommt die Frage auf, ob man irgendwie dafür sorgen kann, dass sich die Intervalle für $d_\alpha - 1$, d_α und $d_\alpha + 1$ zumindest teilweise nicht überschneiden. Wenn Komponenten nach der Feststellung, dass sie $[m]$ -Komponenten sind, einfach n Runden warten, bevor sie das Ende signalisieren, so ergeben sich folgende Veränderungen: wenn $d_\alpha = d - 1$, kann α nach $(b+n)(d-1)$ Runden (also in Runde $(b+n)(d-1) + 1$) schließen, dass sie eine $[m]$ -Komponente ist und wird also in Runde $t_e(d) = (b+n)(d-1) + 1 + n$ das Ende signalisieren. Nun gilt:

$$\begin{aligned}
 t_{e_\alpha}(d_\alpha) &\in \left[\inf(t_s) + \overbrace{(b+n)(d_\alpha - 1) + n + 1}^{t_e(d_\alpha)}, \sup(t_s) + \overbrace{(b+n)(d_\alpha - 1) + n + 1}^{t_e(d_\alpha)} \right] \\
 &= \left[\overbrace{t_{r_\alpha} - (b-1)}^{\inf(t_s)} + \overbrace{(b+n)(d_\alpha - 1) + n + 1}^{t_e(d_\alpha)}, \right. \\
 &\quad \left. \overbrace{t_{r_\alpha} + (b-1)}^{\sup(t_s)} + \overbrace{(b+n)(d_\alpha - 1) + n + 1}^{t_e(d_\alpha)} \right]
 \end{aligned}$$

mit $h := t_{r_\alpha} + (b+n)(d_\alpha - 1) + n + 1$ folgt

$$t_{e_\alpha}(d_\alpha) \in [h - (b-1), h + (b-1)] \quad (3)$$

Setzt man nun $d' := d+1$, wählt also das relevante Intervall für eine weitere existierende $[m]$ -Komponente, so folgt:

$$t_e(d') = t_e(d+1) = (b+n) \cdot (d+1-1) + n + 1 = (b+n)d + n + 1$$

Es folgt also

$$\begin{aligned}
t_{e_\alpha}(d_\alpha) &\in [\inf(t_s) + \overbrace{(b+n)d_\alpha + n + 1}^{t_e(d_\alpha)}, \sup(t_s) + \overbrace{(b+n)d_\alpha + n + 1}^{t_e(d_\alpha)}] \\
&= [\underbrace{t_{r_\alpha} - (b-1)}_{\inf(t_s)} + \overbrace{(b+n)d_\alpha + n + 1}^{t_e(d_\alpha)}, \underbrace{t_{r_\alpha} + (b-1)}_{\sup(t_s)} + \overbrace{(b+n)d_\alpha + n + 1}^{t_e(d_\alpha)}] \\
&= [t_{r_\alpha} - (b-1) + (b+n)d_\alpha + \underbrace{(b+n) - (b+n)}_{=0} + n + 1, \\
&\quad t_{r_\alpha} + (b-1) + (b+n)d_\alpha + \underbrace{(b+n) - (b+n)}_{=0} + n + 1] \\
&= [t_{r_\alpha} - (b-1) + (b+n)(d_\alpha - 1) + (b+n) + n + 1, \\
&\quad t_{r_\alpha} + (b-1) + (b+n)(d_\alpha - 1) + (b+n) + n + 1] \\
&= [\underbrace{t_{r_\alpha} + (b+n)(d_\alpha - 1) + n + 1}_{=h} - (b-1) + (b+n), \\
&\quad \underbrace{t_{r_\alpha} + (b+n)(d_\alpha - 1) + n + 1}_{=h} + (b-1) + (b+n)] \\
&= [h + n + 1, h + 2b + n - 1] \tag{4}
\end{aligned}$$

Gesucht ist eine Methode, mit der man $(3) \cap (4) = \emptyset$ setzt. Betrachten wir dazu das neue Schnittintervall: $(3) \cap (4) = [h + n + 1, h + (b-1)]$. Es ist offensichtlich, dass man die Bedingung $(3) \cap (4) = \emptyset$ erfüllen kann, wenn man n nur gross genug wahlt:

$$\begin{aligned}
h + n + 1 &> h + b - 1 \\
\Leftrightarrow n + 1 &> b - 1 \\
\Leftrightarrow n &> b - 2 \\
\Leftrightarrow n &\geq b - 1 \tag{5}
\end{aligned}$$

Das bringt uns zu *nun bereits bewiesenem*

Satz (b): Eine Komponente kann (jedoch) nur *sicher* schlieen, ob sie eine $[m]$ -Komponente oder eine $\lceil m \rceil$ -Komponente ist, wenn jede Komponente $n \geq b - 1$ Runden wartet, bevor sie das Synchronisationsende signalisiert. In diesem Falle werden *alle* $[m]$ -Komponenten spatestens in Runde $t_e(d) = d(b+n) + 1$ Runden erkannt sein⁴.

2.3 Stark- b -synchrone Kommunikation bei Systemen mit Ausfallen

Stark- b -synchrone Kommunikationssysteme sind hier Systeme, die garantieren, dass jede Nachricht spatestens $(b-1)$ Runden nach Sendung geliefert wird und einen Zeitstempel t_0 der Sendung enthalt. Wenn die Uhren aller Komponenten im System exakt gleich laufen, konnen alle Komponenten dadurch errechnen, wann auf jeden Fall alle Komponenten die Nachricht erhalten haben ($t_s = t_0 + b - 1$). Systeme mit Ausfallen sind hier Systeme, bei denen nicht alle Komponenten so agieren, wie sie designed wurden, d.h., einzelne Komponenten konnen fehlerhaft sein.

Bis auf die *mogliche Fehlerbehaftung* des Systems ist diese Situation aquivalent zur in Sektion 2 als erste vorgestellte Situation. Betrachten wir nochmals den Fall α, β : β sei die einzige $[m]$ -Komponente, die α bekannt ist. Erhalt α nun das Startsignal, so musste nach

⁴also nach $d(b+n)$ Runden

Sektion 2 α nach $t_s + d = t_0 + d = t_0 + 2$ Runden folgern, dass β ebenfalls eine $\lfloor m \rfloor$ -Komponente kennt. Jedoch könnte β defekt sein und deswegen nicht auf die gesendete Nachricht reagiert haben. Das in Sektion 2 vorgestellte Modell ist also für den fehlerbehafteten Fall hinfällig.

Geht man jedoch davon aus, dass es im System Allgemeinwissen ist, welche Komponenten fehlerhaft sind und modifiziert man das Startsignal (bzw. die Startbedingung) zu

- (1) Mindestens eine *funktionierende* Komponente kennt einen niedrigeren Wert als eine andere Komponente.

und (2)-(4) so wie in Sektion 2, so kann man darauf verzichten, die fehlerhaften Komponenten zu betrachten, insofern man davon ausgeht, dass einzelne Komponenten nicht während der Synchronisation ausfallen. Mit diesen Veränderungen hat man Sektion 2 modelliert, folglich ist dann die gleiche Lösung anwendbar.

3 Quick Elimination - eine minimale Lösung

Untersucht man die vorgestellten Algorithmen auf ihre Laufzeit, wird man sehen, dass es sich um Linearzeit-Algorithmen handelt: d $\lfloor m \rfloor$ -Komponenten werden nach d (bzw. $b \cdot d$, also $O(d)$) Runden als solche erkannt. Es bleibt die Frage, ob es einen schnelleren Algorithmus gibt. In [GO84] wurde als Nebenprodukt ein minimaler Konstantzeit-Algorithmus zur Lösung des in Sektion 2 vorgestellten Problems⁵ sichtbar. Dieser $O(1)$ -Algorithmus soll hier nun hergeleitet werden.

Wir gehen davon aus, dass das Startsignal synchron an alle Komponenten geleitet wird, die Komponenten also die Nachricht alle in der gleichen Runde erhalten. Alle Komponenten teilen auch hier ein Speicherbit. Da der Algorithmus eine konstante Zahl an Runden dauern wird, wird das Speicherbit nicht mehr zum Signalisieren des Endes der Synchronisation benötigt. Wir erlauben daher, das Bit mehrfach zu setzen; dazu wird beim Start einer neuen Runde das Bit zurückgesetzt.

Zuerst betrachten wir, wieviele Runden *minimal* benötigt werden, um eine Synchronisation zu vollziehen: Es ist klar, dass, wenn eine Komponente κ von d_κ $\lfloor m \rfloor$ -Komponenten weiß, *jede* Komponente entweder von $d_\kappa - 1$, d_κ oder $d_\kappa + 1$ $\lfloor m \rfloor$ -Komponenten weiß. Ist κ eine $\lfloor m \rfloor$ -Komponente, so gibt es Komponenten, die von $d_\kappa + 1$ wissen, aber keine, die von $d_\kappa - 1$ wissen; ist jedoch κ eine $\lceil m \rceil$ -Komponente, so gibt es Komponenten, die von $d_\kappa - 1$ wissen, aber keine, die von $d_\kappa + 1$ wissen. κ kann sich also eindeutig identifizieren, indem es nach Komponenten λ Ausschau hält mit $d_\lambda = d_\kappa + 1$ oder $d_\lambda = d_\kappa - 1$.

- Wenn nun bei einem beliebigen Algorithmus weder Komponenten α mit $d_\alpha = d_\kappa - 1$ noch Komponenten β mit $d_\beta = d_\kappa + 1$ in der ersten Runde das Bit setzen⁶, kann κ nicht entscheiden, zu welcher Gruppe es selber gehört (ob $d_\kappa = \lfloor m \rfloor$ oder $d_\kappa = \lceil m \rceil$). κ braucht also weitere Informationen aus der 2. Runde, um zu erkennen, ob es α - oder β -Komponenten gibt und eine weitere Runde, um diese Erkenntnis zu nutzen.

⁵Problemstellung: Synchronisation einer Integer-Variable, die zwischen allen Komponenten eines verteilten Systems um maximal 1 abweicht, mit der Bedingung, dass alle Komponenten ein Bit an Speicher teilen und mindestens ein Komponenten-Paar mit unterschiedlichen Werten existiert.

⁶oder beide das Bit setzen

- Wenn dagegen entweder α oder β , aber nicht beide, in der ersten Runde ein Signal geben, kann κ zwar schließen, zu welcher Art es selber gehört, jedoch α bzw. β selber noch nicht; diese brauchen weitere Information von κ in Runde 2, um sich selber in Runde 3 einordnen zu können.

Es werden also bis zu zwei Runden zur Informationssammlung benötigt, um für alle Komponenten sagen zu können, zu welcher Gruppe sie gehören; in der 3. Runde wird diese Erkenntnis dann ggf. noch umgesetzt. Folglich benötigt ein beliebiger Algorithmus **mindestens drei Runden** zur vollständigen Lösung des Problems.

Es bietet sich nun an, den Modulo-Operator zu benutzen, um die unendliche Menge der möglichen Werte für d in die übersichtliche Menge von drei Äquivalenzklassen umzurechnen ($d_\kappa \bmod 3$ für Komponente κ). Auch in den Äquivalenzklassen läßt sich genau unterscheiden, ob eine Komponente $d_\kappa - 1$, d_κ oder $d_\kappa + 1$ $\lfloor m \rfloor$ -Komponenten kennt: $(d_\kappa - 1 \bmod 3) \neq (d_\kappa \bmod 3) \neq (d_\kappa + 1 \bmod 3)$. Eine entsprechende Übersicht aus globaler Sicht findet sich in Abbildung 3.

	$d \bmod 3 = 0$	$d \bmod 3 = 1$	$d \bmod 3 = 2$	
$d_{\lfloor m \rfloor} \bmod 3$	2	0	1	$\lfloor m \rfloor$ -Komponenten
$d_{\lceil m \rceil} \bmod 3$	0	1	2	$\lceil m \rceil$ -Komponenten

Abbildung 3: $d_{\lfloor m \rfloor}$ und $d_{\lceil m \rceil}$ unter Modulo-3-Berücksichtigung.

Lösen wir als erstes den einfachsten Fall: Für $d = 1$, $d_\alpha = 0$ gilt: eine Komponente α , die von $d_\alpha \bmod 3 = 0$ mit $d_\alpha = 0$ $\lfloor m \rfloor$ -Komponenten weiß, kann sich in der ersten Runde zuordnen. Sie setzte in der ersten Runde das geteilte Speicherbit. Zusätzliche sollen *alle* Komponenten α das Bit setzen, für die gilt: $d_\alpha \bmod 3 = 0$ (auch, wenn sich diese bisher nicht zuordnen können).

Wurde in der ersten Runde das Bit gesetzt, so gibt es also Komponenten α mit $d_\alpha \bmod 3 = 0$. Komponenten β mit $d_\beta \bmod 3 = (d_\alpha - 1) \bmod 3 = 2$ können also schließen, dass sie $\lfloor m \rfloor$ -Komponenten sind. Sie setzen in der 2. Runde das Bit. Wenn die α -Komponenten dies sehen, können sie in Runde drei schließen, dass sie selber $\lceil m \rceil$ -Komponenten sind.

Wurde das Bit in der ersten, aber *nicht* in der zweiten Runde gesetzt, so gibt es also Komponenten γ mit $d_\gamma \bmod 3 = (d_\alpha + 1) \bmod 3 = 1$. α -Komponenten können sich dann wegen des fehlenden Signals in der zweiten Runde spätestens in der dritten Runde als $\lfloor m \rfloor$ -Komponenten identifizieren.

Wurde in der ersten Runde *nicht* das Bit gesetzt, so gibt es keine Komponenten α mit $d_\alpha \bmod 3 = 0$. Es kann also nur Komponenten δ mit $d_\delta \bmod 3 = 1$ und Komponenten ϵ mit $d_\epsilon \bmod 3 = 2$ geben. Offensichtlich sind die δ -Komponenten $\lfloor m \rfloor$ -Komponenten und die ϵ -Komponenten $\lceil m \rceil$ -Komponenten und beide können dies in der zweiten Runde schließen. Die δ -Komponenten setzen dann in der zweiten Runde das Bit.

Für den Fall, dass weder in der ersten Runde, noch in der zweiten Runde das Bit gesetzt

wurde, gilt: es gibt weder α -Komponenten mit $d_\alpha \bmod 3 = 0$ noch gibt es δ -Komponenten mit $d_\delta \bmod 3 = 1$. Folglich sind alle Komponenten im System ϵ -Komponenten. Daraus folgt, dass es nur $\lfloor m \rfloor$ -Komponenten oder nur $\lceil m \rceil$ -Komponenten gibt. Da nach Voraussetzung dieser Fall nicht auftreten kann, spielt er in diesem Algorithmus eigentlich keine Rolle. Adaptiert man den Algorithmus jedoch z.B. auf das Rätsel Mamajorcas, so ist möglich, daß *alle* Frauen von ihren Männern betrogen werden. Damit dieser Fall über den von der Königin aufgestellten Regeln korrekt terminiert, wird nun angenommen, dass wenn $\{\alpha \mid d_\alpha \bmod 3 = 0\} = \emptyset$ und $\{\delta \mid d_\delta \bmod 3 = 1\} = \emptyset$, *alle* Komponenten $\lfloor m \rfloor$ -Komponenten sind. Wird also das Bit weder in der ersten Runde noch in der zweiten Runde gesetzt, so können sich *alle* Komponenten in der dritten Runde als $\lfloor m \rfloor$ -Komponenten identifizieren. Äquivalentes Verhalten ergibt sich automatisch aus den bisherigen Regeln, wenn es nur $\lfloor m \rfloor$ -Komponenten κ gibt mit $d_\kappa \bmod 3 = 0$ oder mit $d_\kappa \bmod 3 = 1$.

Fasst man den Algorithmus etwas kompakter zusammen, so ergibt sich:

- R1: Alle Komponenten κ , die von $d_\kappa : d_\kappa \bmod 3 = 0$ $\lfloor m \rfloor$ -Komponenten wissen, setzen in der ersten Runde das geteilte Speicherbit. Wenn $d_\kappa = 0$, ist κ eine $\lfloor m \rfloor$ -Komponente.
- R2: (a) Wenn in der ersten Runde das Bit *nicht* gesetzt wurde, sind alle Komponenten κ $\lfloor m \rfloor$ -Komponenten, die $d_\kappa : d_\kappa \bmod 3 = 1$ $\lfloor m \rfloor$ -Komponenten kennen. Diese Komponenten κ setzen das Speicherbit.
 (b) Wenn jedoch in der ersten Runde das Bit gesetzt wurde, sind alle Komponenten κ $\lfloor m \rfloor$ -Komponenten, die $d_\kappa : d_\kappa \bmod 3 = 2$ $\lfloor m \rfloor$ -Komponenten kennen. Diese Komponenten κ setzen das Bit.
- R3: (a) Wenn weder in Runde 1 noch in Runde 2 das Bit gesetzt wurde, sind *alle* Komponenten $\lfloor m \rfloor$ -Komponenten.
 (b) Wenn in der ersten Runde das Bit gesetzt wurde, aber nicht in der zweiten, so sind alle Komponenten $\lfloor m \rfloor$ -Komponenten, die in der ersten Runde das Bit gesetzt haben.

Zur Probe überprüfen wir die **Korrektheit** des angegebenen Algorithmus: Sei d die Anzahl der $\lfloor m \rfloor$ -Komponenten. Bedingung zur Anwendung des Algorithmus ist natürlich, dass es mindestens eine $\lfloor m \rfloor$ -Komponente gibt: $d \geq 1$ (deshalb verzichten wir an dieser Stelle auf eine Überprüfung für $K_{\lceil m \rceil} = \emptyset$). Es gibt dann immer die nicht-leere Menge $K_{\lfloor m \rfloor}$ aus $\lfloor m \rfloor$ -Komponenten, die $d_{\lfloor m \rfloor} = d - 1$ $\lfloor m \rfloor$ -Komponenten kennen und die nicht-leere Menge $K_{\lceil m \rceil}$ aus $\lceil m \rceil$ -Komponenten, die $d_{\lceil m \rceil} = d$ $\lfloor m \rfloor$ -Komponenten kennen.

- Für $d \bmod 3 = 0$ gilt: $d_{\lceil m \rceil} \bmod 3 = 0$, $d_{\lfloor m \rfloor} \bmod 3 = 2$.

Komponenten aus $K_{\lceil m \rceil}$ werden in der ersten Runde das Speicherbit setzen. Da es aber mehr als nur eine $\lfloor m \rfloor$ -Komponente gibt ($d \in \{3 \cdot n \mid n \in \mathbb{N}\}$), wird keine Komponente falsch erkannt. In der 2. Runde werden dann alle Komponenten aus $K_{\lfloor m \rfloor}$ als solche erkannt und setzen das Speicherbit. In Runde 3 passiert daher nichts.

- Für $d \bmod 3 = 1$ gilt: $d_{\lceil m \rceil} \bmod 3 = 1$, $d_{\lfloor m \rfloor} \bmod 3 = 0$.

$\lfloor m \rfloor$ -Komponenten setzen in der ersten Runde das Speicherbit. Da in Runde 2 nichts passiert (denn in Runde 1 wurde das Bit gesetzt), werden alle $\lfloor m \rfloor$ -Komponenten in Runde 3 als solche erkannt.

- Für $d \bmod 3 = 2$ gilt: $d_{\lceil m \rceil} \bmod 3 = 2$, $d_{\lfloor m \rfloor} \bmod 3 = 1$.

In der ersten Runde passiert nichts. Deswegen werden in Runde 2 alle $\lfloor m \rfloor$ -Komponenten unmittelbar erkannt. In Runde 3 passiert daher nichts mehr.

Der vorgestellte Algorithmus ist also minimal, da er das Problem in maximal drei Runden löst, und er ist korrekt.

4 Ergebnisse

In [HM84] wird unter anderem gezeigt, dass das Erreichen von Allgemeinwissen in verteilten Systemen nicht möglich ist, wenn *verlustlose Kommunikation mit einer Zeitgrenze*, *Ausfallsicherheit* oder *exakte Synchronisation* nicht garantiert sind; desweiteren wurden verschiedene Arten von „Allgemeinwissen“ entwickelt, die dennoch in solchen Fällen erreichbar sind.

Hier (und ursprünglich in [MDH85]) wurden nun einige Beispiele aufgezeigt, um zum einen das Versagen von Algorithmen zu demonstrieren, die auf Allgemeinwissen basieren, wenn sie in solchen Systemen genutzt werden; auf der anderen Seite werden Sonderfälle (wie 2.2) aufgezeigt, in denen Algorithmen dennoch teilweise funktionieren (weil die Grundvoraussetzungen über Umwege konstruiert werden können). Auch wenn der vorgestellte Algorithmus für oben genannte verteilte Systeme kaum nutzbar erscheint, so ist er ein gutes Beispiel für die Notwendigkeit, äußerste Vorsicht beim Design von Algorithmen für verteilte Systeme walten zu lassen.

Insgesamt wird klar, dass eine **Modellierung von verteilten Systemen durch Wissen einzelner Komponenten im System, Kommunikation von Komponenten untereinander und Aktion einzelner Komponenten** vielversprechend ist.

Abbildungsverzeichnis

1	Zwei verschiedene Erklärungen für das Verhalten von β	7
2	Sich überschneidende Intervalle für d_α und $d_\alpha + 1$	9
3	$d_{\lceil m \rceil}$ und $d_{\lfloor m \rfloor}$ unter Modulo-3-Berücksichtigung.	12

Literatur

- [GO84] Abbas El Gamal and Alon Orlitsky. Interactive data compression. *25th Annual Symposium on Foundations of Computer Science*, pages 100–108, 1984.
- [HM84] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. In *PODC '84: Proceedings of the third annual ACM*

symposium on Principles of distributed computing, pages 50–61, New York, NY, USA, Oct. 1984. ACM Press.

- [MDH85] Yoram Moses, Danny Dolev, and Joseph Y. Halpern. Cheating husbands and other stories (preliminary version): a case study of knowledge, action, and communication. In *PODC '85: Proceedings of the fourth annual ACM symposium on Principles of distributed computing*, pages 215–223, New York, NY, USA, 1985. ACM Press.
- [Tel00] Gerard Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 2000. ISBN 0521794838.