ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

HS 2016 Prof. Dr. Roger Wattenhofer

Prüfung
Verteilte Systeme
Teil 2

Freitag, 10. Februar 2017
9:00 – 12:00

Die Anzahl Punkte pro Teilaufgabe steht jeweils in Klammern bei der Aufgabe. Sie dürfen die Fragen auf Englisch oder auf Deutsch beantworten. **Begründen Sie alle Ihre Antworten und beschriften Sie Skizzen und Zeichnungen verständlich.** Schreiben Sie zu Beginn Ihren Namen und Ihre Legi-Nummer in das folgende dafür vorgesehene Feld.

Name	Legi-Nr.

Frage Nr.	Erreichte Punkte	Maximale Punkte
8		12
9		16
10		17
11		20
12		25
Total		90

8 Multiple Choice (12 Punkte)

Beurteilen Sie, ob die folgenden Aussagen richtig oder falsch sind, und kreuzen Sie die entsprechenden Felder an. Eine richtig beurteilte Aussage gibt 1 Punkt, eine nicht beurteilte Aussage 0 Punkte, eine nicht richtig beurteilte Aussage **-1 Punkt**. Die **gesamte** Aufgabe wird mit minimal 0 Punkten bewertet.

A) Eventual Consistency & Bitcoin

Aussage	**Wahr**	**Falsch**
Addiert man die Werte aller UXTOs auf, so erhält man die Menge der bis dato gemünzten (mined) Bitcoins.	☐	☐
Vergäbe eine neutrale Zentrale fortlaufende Transaktionsnummern, könnte Bitcoin-Doublespends vorgebeugt werden.	☐	☐
Wenn eine Transaktion nicht in den nächsten Block aufgenommen wird, muss sie neu signiert werden, da sie den Hash des letzten Blocks enthält.	☐	☐
Damit ein Angreifer den Zustand des Bitcoin-Netzwerks beliebig manipulieren kann, muss es $f \geq n/2$ byzantinische Knoten im Netzwerk geben.	☐	☐

B) Distributed Storage

Aussage	**Wahr**	**Falsch**
In einer DHT mit Consistent Hashing ist immer derselbe Knoten für einen Schlüssel zuständig, auch mit Churn.	☐	☐
Der Hauptvorteil von Consistent Hashing gegenüber Linearem Hashing ($h = hash(O) \mod n$, wobei O ein Objekt und n die Anzahl Knoten sind) ist, dass man bei Consistent Hashing beliebige Hypergraphen als Overlay Graph verwenden kann.	☐	☐
Der Hypercube ist am besten geeignet als Overlay Netzwerk für eine DHT.	☐	☐
Der Durchmesser eines Gittergraphen mit $n = m * m$ Knoten ist $m - 1$	☐	☐

C) Consistency & Transactional Memory

Aussage	**Wahr**	**Falsch**
Causal Consistency impliziert Sequential Consistency.	☐	☐
Eine Ausführung ist linearisierbar wenn sich keine zwei Aufrufe zeitlich überlappen.	☐	☐
Quiescent Consistency macht keine Aussage darüber, ob ein ausgeführtes Programm *wait-free* ist.	☐	☐
Auf einem 64-bit-System ermöglicht Transactional Memory es, 9 Speicherzellen atomar zu verändern.	☐	☐

9 Lesezugriffe in Zyzzyva (16 Punkte)

In Zyzzyva sortiert ein Primary sämtliche Kommandos, die von Clients angefragt werden. Operationen, die den Zustand der Replicas verändern, müssen eine eindeutige zeitliche Abfolge haben, um zu verhindern, dass der Zustand der replizierten state machine divergiert. In dieser Frage untersuchen Sie, ob dies auch für reine Lesezugriffe unbedingt notwendig ist.

Wir ändern Zyzzyva so, dass ein Client u für reine Lesezugriffe direkt alle Replicas r mittels einer $\texttt{R} = \texttt{Read-Only-Request}(c)_u$ Nachricht anfragt. Korrekte Replicas beantworten diese Requests direkt mit $\texttt{Read-Only-Response}(\text{R},a,h^r)_r$ wobei c das angefragte Kommando ist, a ist das Resultat des Lesezugriffs und h^r ist die lokale history des antwortenden Replicas. Die in den Antworten enthaltenen lokalen histories h^r beinhalten keine commit Zertifikate. Diese Lesezugriffe werden ausserdem nicht in der lokalen history vermerkt.

Annahme: Das System besteht aus 7 Replicas (inklusive Primary), wovon höchstens 2 byzantinisch sein können. Ausserdem können Sie annehmen, dass alle histories (h^r) und alle Antworten (a) in allen erhaltenen Nachrichten konsistent/gleich sind für die nachfolgenden Fragen.

Definition (korrekt). *Wir bezeichnen die Antwort auf ein Kommando als **korrekt**, falls sich in Zyzzyva die history vor diesem Kommando unter keinen Umständen ändern wird/kann.*

A) [8] Nehmen Sie an, dass der Client 7 signierte Antworten von 7 Replicas erhält. Kann der Client sicher sein, dass die Antwort **korrekt** ist?

B) [8] Nehmen Sie an, dass der Client nur 5 signierte Antworten von 5 Replicas erhält. Welche zusätzlichen Informationen müssten Replicas diesen Antworten anfügen damit der Client sicher stellen kann, dass der Lesezugriff **korrekt** ist?

10 k-Artikel-Auktion (17 Punkte)

A) [4] Ist eine Auktion, bei der der Gewinner den dritthöchsten Preis zahlen muss, wahrheitsgemäss (truthful)?

Es sollen $k > 1$ Artikel an $n > k$ Bieter versteigert werden. Alle k Artikel sind identisch, und jeder Bieter möchte nur einen Artikel erstehen. Jeder Bieter i hat eine geheime Wertschätzung v_i für jeden Artikel.

B) [8] Eine Möglichkeit ist, für jeden der k Artikel nacheinander eine Second-Price-Auktion durchzuführen. Ist das ein wahrheitsgemässer Mechanismus? Wenn ja, beweisen Sie es. Wenn nein, geben Sie ein Gegenbeispiel an *und* finden Sie einen wahrheitsgemässen Mechanismus.

Nun betrachten wir eine Variation dieses Szenarios: die k Artikel sind verschieden und jeder Bieter i hat eine additive Wertschätzung, d.h., für ein Bündel von Artikeln S ist $v_i(S) := \sum_{j \in S} v_{ij}$, wobei v_{ij} die geheime Wertschätzung von Bieter i für Artikel j ist. Ein Bieter ist nun also bereit, mehrere Artikel zu erstehen.

C) [5] Entwerfen Sie einen wahrheitsgemässen Mechanismus für dieses Szenario.

11 Locking & Concurrency (20 Punkte)

Als Lock für ein faires Multithread-System soll eine FIFO Queue zum Einsatz kommen, die als doppelt verlinkte Liste implementiert ist. Die Liste hast einen *head*- und einen *tail*-Block, die vor dem ersten, beziehungsweise nach dem letzten, Element der Liste stehen. Um sich in die Warteschlange einzureihen, können Threads einen Knoten am Ende der Liste (vor dem tail) anhängen. Jeweils der Thread, von dem der vorderste Knoten (nach dem head) stammt, hat das Lock. Wenn er fertig ist, löscht er den Knoten, wodurch der nächste Knoten nachrückt, in dem er erkennt, dass er der vorderste ist.

A) [3] Ist es besser, wenn die Threads prüfen, ob ihr Knoten auf den head der Liste folgt, oder wenn der Thread mit dem Lock jeweils den folgenden Knoten freigibt, wenn er fertig ist, indem er ein Flag in dem nachfolgenden Knoten setzt?

B) [7] Die Idee ist in folgendem Pseudocode umgesetzt. Es wird angenommen, dass alle Reads und Writes atomar sind. Wieso ist die Implementierung nicht korrekt? Mit welcher Strategie kann das Problem gelöst werden?

```
1:  Struct Block {
2:      Boolean locked
3:      Pointer next
4:      Pointer previous
5:  }
6:
7:  Block head, tail
8:  head.locked = false
9:  head.next = tail
10: tail.locked = false
11: tail.previous = head
12:
13: procedure ACQUIRE_LOCK
14:     do
15:         res = TestAndSet(tail.locked)
16:     while res = true
17:
18:     Block prev = tail.previous
19:     do
20:         res = TestAndSet(prev.locked)
21:     while res = true
22:
23:     Block new_block
24:     prev.next = new_block
25:     new_block.prev = prev
26:     tail.prev = new_block
27:
28:     tail.locked = false
29:     prev.locked = false
30:     return new_block
31: procedure Thread implementation
32:     mine = ACQUIRE_LOCK()
33:     while head.next != mine do
34:         wait 10 ms
35:     [Critical Section]
36:     RELEASE_LOCK()
37:
38: procedure RELEASE_LOCK
39:     do
40:         res = TestAndSet(head.locked)
41:     while res = true
42:
43:     Block mine = head.next
44:     do
45:         res = TestAndSet(mine.locked)
46:     while res = true
47:
48:     Block next = mine.next
49:     do
50:         res = TestAndSet(next.locked)
51:     while res = true
52:
53:     head.next = next
54:     next.previous = head
55:
56:     head.locked = false
57:     mine.locked = false
58:     next.locked = false
```

C) [7] Ist diese Implementierung korrekt wenn die Liste immer mehr als 100 Blöcke enthält? Zeigen Sie wieso oder geben Sie ein Gegenbeispiel.

D) [3] Welches Performanceproblem kann in dieser Implementierung auftreten?

12 Erdbeben-Synchronisierung (25 Punkte)

In einem verteilten Netzwerk soll die Zeit von sechs Knoten synchronisiert werden. Da alle Knoten einen Beschleunigungsmesser haben, möchte man ein Erdbeben zum Synchronisieren verwenden. Die Knoten können Daten untereinander kommunizieren mit stark variierenden Übertragungsverzögerungen. Die Knoten haben alle eine exakte Hardwareclock ohne Drift, jedoch einen konstanten Zeitoffset zueinander.

A) [6] Zwei Ingenieure überlegen sich zwei Synchronisierungsalgorithmen. Der erste Ingenieur schlägt vor, die Uhr eines Knotens auf 0 zu setzen wenn das Erdbeben am Knoten gemessen wird. Der zweite schlägt vor, dass jeder Knoten p_i die lokale Zeit t_i des Erdbebens an alle anderen Knoten mitteilt. Danach berechnet jeder Knoten p_i den Median aller gemessenen Zeiten $\tilde{t}$ des Erdbebens und korrigiert seine Uhr um $\tilde{t} - t_i$. Welcher Algorithmus synchronisiert besser? Begründen Sie.

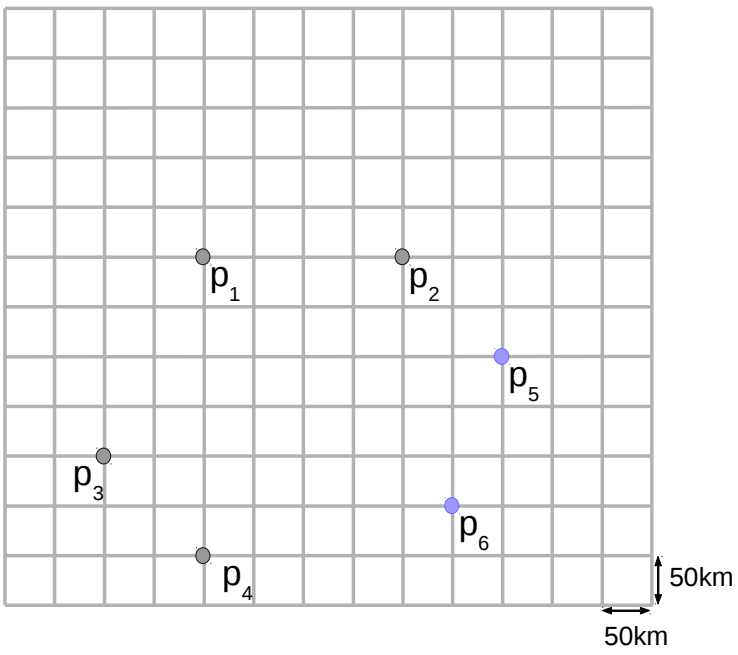

Abbildung 1: Anordnung der Knoten.

Beide vorgeschlagenen Algorithmen sind noch zu ungenau. Nun möchte man eine genaue Synchronisierung erreichen. Von den Knoten p_1, p_2, p_3 und p_4 ist bekannt, dass sie bereits synchronisiert sind. Wir nehmen an, dass sich der Ursprung des Erdbebens und die sechs Knoten in einer Ebene befinden. Die Positionen der Knoten sind in Abbildung 1 gegeben. Die Ausbreitungsgeschwindigkeit der seismischen Wellen beträgt $50km/s$. Die Knoten detektieren das Erdbeben zu den jeweiligen lokalen Zeiten $t_1 = 200s$, $t_2 = 200s$, $t_3 = 201.64s$, $t_4 = 201.64s$, $t_5 = 306s$, $t_6 = 402s$. Der Gitterabstand beträgt $50km$.

B) `[9]` Wie können die restlichen zwei Knoten synchronisiert werden? Benutzen Sie die Skizze in Abbildung 1. Um wie viel müssen die Knoten p_5 und p_6 ihre Uhren korrigieren, damit sie auch synchron zu den restlichen vier sind?

C) `[4]` Nehmen Sie an, dass vier der sechs Knoten synchronisiert sind. Man weiss jedoch nicht, welche vier. Wie können Sie mit Hilfe eines einzigen Erdbebens die synchronisierten Knoten bestimmen?

D) [6] Ist die Synchroniserung auch möglich wenn alle sechs Knoten nicht synchronisiert sind? Wie viele Erdbeben müssten für die Synchronierung mindestens gemessen werden?

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

HS 2016 Prof. Dr. Roger Wattenhofer

Prüfung
Verteilte Systeme
Teil 2

Freitag, 10. Februar 2017
9:00 – 12:00

Die Anzahl Punkte pro Teilaufgabe steht jeweils in Klammern bei der Aufgabe. Sie dürfen die Fragen auf Englisch oder auf Deutsch beantworten. **Begründen Sie alle Ihre Antworten und beschriften Sie Skizzen und Zeichnungen verständlich.** Schreiben Sie zu Beginn Ihren Namen und Ihre Legi-Nummer in das folgende dafür vorgesehene Feld.

Musterlösung

Diese Musterlösung enthält für jede Aufgabe eine beispielhafte Lösung, die die volle Punktzahl erreicht hätte. Andere Lösungswege konnten jedoch ebenfalls ein Teil oder alle der Punkte erhalten.

Frage Nr.	Erreichte Punkte	Maximale Punkte
8		12
9		16
10		17
11		20
12		25
Total		90

8 Multiple Choice (12 Punkte)

Beurteilen Sie, ob die folgenden Aussagen richtig oder falsch sind, und kreuzen Sie die entsprechenden Felder an. Eine richtig beurteilte Aussage gibt 1 Punkt, eine nicht beurteilte Aussage 0 Punkte, eine nicht richtig beurteilte Aussage **-1 Punkt**. Die **gesamte** Aufgabe wird mit minimal 0 Punkten bewertet.

A) Eventual Consistency & Bitcoin

Aussage	**Wahr**	**Falsch**
Addiert man die Werte aller UXTOs auf, so erhält man die Menge der bis dato gemünzten (mined) Bitcoins. *Reason:*	✓	
Vergäbe eine neutrale Zentrale fortlaufende Transaktionsnummern, könnte Bitcoin-Doublespends vorgebeugt werden. *Reason: Ordering = Agreement on validity*	✓	
Wenn eine Transaktion nicht in den nächsten Block aufgenommen wird, muss sie neu signiert werden, da sie den Hash des letzten Blocks enthält. *Reason: Nein, die Transaktion bleibt gültig und wird früher oder später von selbst aufgenommen.*		✓
Damit ein Angreifer den Zustand des Bitcoin-Netzwerks beliebig manipulieren kann, muss es $f \geq n/2$ byzantinische Knoten im Netzwerk geben. *Reason: Wegen Proof of Work kommt es nicht auf die Anzahl sondern auf die summierte Rechenkraft an.*	✓	

B) Distributed Storage

Aussage	**Wahr**	**Falsch**
In einer DHT mit Consistent Hashing ist immer derselbe Knoten für einen Schlüssel zuständig, auch mit Churn. *Reason: Wenn der Knoten wegfällt muss ein anderer diesen übernehmen. Dasselbe wenn ein Knoten näher beim Schlüssel auftaucht.*		✓
Der Hauptvorteil von Consistent Hashing gegenüber Linearem Hashing ($h = hash(O) \mod n$, wobei O ein Objekt und n die Anzahl Knoten sind) ist, dass man bei Consistent Hashing beliebige Hypergraphen als Overlay Graph verwenden kann. *Reason: Consistent und Linear Hashing können die selben Hypergraphen verwenden. Der Hauptvorteil ist, dass bei Consistent Hashing beliebig Knoten hinzukommen/verschwinden können mit minimalem Aufwand. Bei Linearem Hashing müssen jedesmal die Hälfte der Keys umverteilt werden, da die Zuteilung Abhängig von der Anzahl Knoten ist.*		✓
Der Hypercube ist am besten geeignet als Overlay Netzwerk für eine DHT. *Reason: Welchen Hypergraphen man verwendet hängt von der Anwendung ab. Es gibt verschiedene Trade-Offs zwischen Durchmesser, Homogenität, etc.*		✓
Der Durchmesser eines Gittergraphen mit $n = m * m$ Knoten ist $m - 1$ *Reason:* $2 * m$		✓

C) Consistency & Transactional Memory

Aussage	**Wahr**	**Falsch**
Causal Consistency impliziert Sequential Consistency. *Reason: Sequential Consistency impliziert Causal Consistency.*		✓
Eine Ausführung ist linearisierbar wenn sich keine zwei Aufrufe zeitlich überlappen. *Reason: Das ist hinreichend, wenn auch nicht notwendig.*	✓	
Quiescent Consistency macht keine Aussage darüber, ob ein ausgeführtes Programm *wait-free* ist. *Reason: Korrekt.*	✓	
Auf einem 64-bit-System ermöglicht Transactional Memory es, 9 Speicherzellen atomar zu verändern. *Reason: Jop. Beliebig viele Zellen sogar.*	✓	

9 Lesezugriffe in Zyzzyva (16 Punkte)

A) If the histories and responses are all the same for every replica, we can be sure that no preceding command is dropped from the history. Commands are only dropped when the view is changed in which case commands reported by f+1 replicas are carried over (plus whatever happened before the last commit certificate). This means that in the case of 3f+1 consistent responses, all correct replicas answered based on a history that does not contain commands that might be dropped.

B) If the 2f+1 responses contain consistent histories which end in a commit certificate the client can be sure. So the client has to include the last commit certificate into the response.

10 k-Artikel-Auktion (17 Punkte)

A) Nein. Gegenbeispiel: alle Wertschätzungen sind verschieden und jeder bietet seine Wertschätzung. In dieser Situation ist es für den Bieter mit der zweithöchsten Wertschätzung vorteilhaft, mehr als das bisherige höchste Gebot zu bieten.

B) Nein, das ist kein wahrheitsgemässer Mechanismus. Gegenbeispiel: alle Wertschätzungen sind verschieden und jeder bietet jede Runde seine Wertschätzung (0, falls er bereits einen Artikel erstanden hat). In dieser Situation ist es für den Bieter mit der höchsten Wertschätzung vorteilhaft, in den ersten Runden ein niedriges Gebot abzugeben, so dass er vorerst keine Auktion gewinnt. Wenn er dann später seine Wertschätzung bietet, gewinnt er einen Artikel, aber zahlt einen tieferen Preis als er in der ersten Runde gezahlt hätte, da in der Zwischenzeit die Bieter mit den nächsthöchsten Wertschätzungen bereits Artikel erhalten haben und ausgestiegen sind.

Wahrheitsgemässer Mechanismus: nur eine Auktionsrunde, in der die k höchsten Gebote die k Artikel jeweils zum Preis des $k+1$-höchsten Gebots erstehen.

C) Nun funktioniert der in B) vorgeschlagene Mechanismus: man führt für jeden der k Artikel eine separate Second-Price-Auktion durch.

11 Locking & Concurrency (20 Punkte)

A) Die zweite Variante (wie MCS Lock) ist besser. Die erste invalidiert die Caches von allen Prozessoren, immer wenn der HEAD geändert wird. Das braucht hohe Bandbreite vom gemeinsamen Memory Bus.

B) In der Implementierung können Deadlocks entstehen: Release locks from front, acquire from back. If queue contains only one block and both operations are issued concurrently, a deadlock can happen when the release thread locks head and the only block and the acquire thread locks the tail and then both wait for each other to release a lock.

Solution: Acquire locks in the same order.
Correct code (not required to get points):

```
1: procedure WAIT_FOR_LOCK
2:     prev = tail.previous
3:     lock(prev)
4:     lock(tail)
5:     if tail.previous != prev then repeat from beginning
6:     [critical section]
```

C) Die Lösung ist korrekt: Bei beiden Methoden kann immer nur ein Thread tail.locked, bzw. head.locked auf true setzen und false als Rückgabewert kriegen. Deshalb müssen dann alle anderen Threads warten, bis tail, bzw. head, wieder "frei" sind. Überholen innerhalb einer Methode ist also nicht möglich. Zwischen den beiden Methoden ist auch keine Beeinflussung möglich, da immer die Blöcke vor und hinter der zu bearbeitenden Position "gesperrt" werden durch das setzen der locked Flags.

D) Das Warten in der Thread-Implementierung ist das Problem. Der Thread, der an der Reihe ist, kann noch bis zu (knapp unter) 10 ms warten, bis er den kritischen Abschnitt ausführt. Währenddessen macht kein einziger Thread Fortschritt.

12 Erdbeben-Synchronisierung (25 Punkte)

A) Beide Algorithmen sind genau gleich gut. Der erste Algorithmus erreicht, dass alle Knoten die Zeit auf 0 setzen wenn das Erdbeben gemessen wird. Der zweite Algorithmus korrigiert die Zeit so, dass alle Uhren $\tilde{t}$ angezeigt hätten als das Erdbeben gemessen wurde.

B) Das Erbeben ist von p_1 und p_2 gleich weit entfernt, also auf einer Gerade. Auch gleich weit von p_3 und p_4. Ursprung befindet sich in Schnittpunkt der zwei Geraden. Von p_1 aus kann die Zeit des Erdbebens berechnet werden. Die korrekte Zeit bei p_5 und $_6$ kann mit der Laufzeitdifferenz vom Ursprung des Erdbebens berechnet werden. Zeit des Erdbebens $t = 200s - 2\sqrt{2}s \approx 197.17s$. Das Erdbeben kam also um $t_{5,corr} = t + 4s \approx 201.17s$ und $t_{6,corr} = t + 3\sqrt{2} \approx 201.41s$ bei den Knoten p_5 und p_6 an. Sie müssen also ihre Zeit um $200s - 2\sqrt{2}s + 4s - 306s = -102s - 2\sqrt{2}s \approx -104.83s$ und $200s - 2\sqrt{2}s + 3\sqrt{2}s - 402s = -202s + \sqrt{2}s \approx -200.59s$ anpassen.

C) Die Position des Erdbebens kann mit 3 synchronisierten Knoten bestimmt werden. Mit vier Knoten ist das Gleichungssystem überbestimmt. Man kann für alle Subsets von 4 Knoten die Position des Erdbebens berechnen. Nur wenn das überbestimmte Gleichungssystem eine Lösung hat, können die 4 Knoten synchron sein.

D) Unbekannt sind der Ursprung der Erdbeben ($2n$ Unbekannte), die Zeit jedes Erdbebens (n Unbekannte) und die Zeitoffsets jedes Knotens zu einem Knoten (5 Unbekannte). Jedes Erdbeben gibt eine Gleichung für jeden Knoten. $5 + 3n \leq 6n$, also mindestens 2 Erdbeben.