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.