ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

HS 2013 Prof. Dr. Roger Wattenhofer

Prüfung
Verteilte Systeme
Teil 2

Freitag, 7. Februar 2014
9:00 – 12:00

Die Anzahl Punkte pro Teilaufgabe steht jeweils in Klammern bei der Aufgabe. Sie dürfen die Fragen englisch oder 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.

Punkte

Frage Nr.	Erreichte Punkte	Maximale Punkte
9		30
10		15
11		15
12		14
13		8
14		8
Total		90

9 Multiple Choice (30 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) Consensus & Consistency

Aussage	**Wahr**	**Falsch**
Mit Compare&Swap kann man Test&Set emulieren.	☐	☐
PBFT setzt authentifizierte Nachrichten voraus.	☐	☐
In einem System mit n Prozessen hat jedes Quorum mindestens $n/2$ Prozesse.	☐	☐
Paxos garantiert keine Liveness (Fortschritt).	☐	☐
In einem Quorum-System ist jeder Prozess Teil von mindestens 2 Quoren.	☐	☐
Die *resilience* eines Quorum-Systems ist maximal die Grösse des kleinsten Quorums.	☐	☐
Ein wait-free Algorithmus darf Locks nur verwenden, wenn keiner der Prozesse byzantinisch ist.	☐	☐
Wenn man zwei *RMW* Operationen mit Konsens-Nummer 2 miteinander kombiniert, bekommt man eine *RMW* Operation mit Konsens-Nummer 4.	☐	☐
Es existiert eine *RMW* Operation mit Konsens-Nummer genau 5.	☐	☐
Wenn ein Protokoll kausale Konsistenz garantiert, ist jede Ausführung linearisierbar.	☐	☐

B) Locking

Aussage	**Wahr**	**Falsch**
Bei optimistischer Synchronisierung auf einer einfach verketteten Liste braucht man nach dem lockfreien Finden des gesuchten Knotens kein Hand-over-Hand-Locking mehr anzuwenden.	☐	☐
Ein faires Queue-Lock kann mittels einer atomaren Test-And-Set-Instruktion implementiert werden.	☐	☐
Locking ist nur auf Mehrkern-Systemen notwendig.	☐	☐
Wenn ein Thread weiter nützliche Arbeit verrichtet, während er auf ein Lock wartet, nennt man das „busy-waiting“.	☐	☐
Test-Test-And-Set-Locks leiden unter dem Problem, dass alle wartenden Prozessoren zugleich auf den Bus zugreifen, wenn das Lock freigegeben wird.	☐	☐
Anderson-Queue-Locks sind fair.	☐	☐
Mutual Exclusion bedeutet, dass immer genau ein Prozess in der Critical Section ist.	☐	☐
Anderson-Queue-Locks leiden unter dem Problem, dass alle wartenden Prozessoren zugleich auf den Bus zugreifen, wenn das Lock freigegeben wird.	☐	☐
Sentinelknoten besitzen mehr Pointer als normale Knoten.	☐	☐

C) Game Theory

Aussage	**Wahr**	**Falsch**
Jedes dominante Strategieprofil ist ebenfalls ein Nash-Gleichgewicht.	☐	☐
In jedem reinen Nash-Gleichgewicht erzielt mindestens ein Spieler seinen maximalen Nutzen.	☐	☐
Jedes Spiel hat ein reines Nash-Gleichgewicht.	☐	☐
In der Second-Price-Auction gereicht es den Bietern nie zum Nachteil die Wahrheit zu sagen.	☐	☐
Der Price Of Anarchy kann nie grösser sein als n.	☐	☐

D) Clock Sync

Aussage	**Wahr**	**Falsch**
FTSP Clock Synchronisation hat einen kleineren lokalen als globalen Skew.	☐	☐
Ist die genaue Round-Trip-Time zwischen Client und Server bekannt, kann auch die Zeit zwischen den beiden Rechnern exakt synchronisiert werden.	☐	☐
Die Abkürzung UTC ist aus dem Englischen abgeleitet und wurde von Craig Venter eingeführt.	☐	☐
Falls ein Clock Synchronisation-Algorithmus A einen kleineren lokalen Skew als Algorithmus B erreicht, dann wird Algorithmus A im Vergleich zu Algorithmus B auch einen kleineren globalen Skew erreichen.	☐	☐

E) Parallel Programming

Aussage	**Wahr**	**Falsch**
Mit OpenMP lässt sich jede **`for`**-Schleife problemlos parallelisieren, indem Sie mit **`#pragma`** `omp parallel` **`for`** eingeleitet wird.	☐	☐
Wenn es bei t Threads $k \cdot t$ Arbeitspakete gibt und OpenMP mit der `schedule(`**`static`**`)`-Direktive verwendet wird, dann werden jedem Thread gleich viele zugeteilt.	☐	☐

10 Consensus (15 Punkte)

Die Neutral Swiss Agency (NSA) ist ein Schweizer Geheimdienst, der die virtuellen Rechte der Bürger überwacht. Vor kurzem wurde entschieden, dass es im nationalen Interesse ist, die Kommunikation aller Erdbewohner zu kontrollieren. Die NSA kann Nachrichten, die über Verbindungen (edges) in der Schweiz gehen, beliebig verändern oder löschen. Die Endpunkte einer Verbindung können nicht wissen, ob diese durch die Schweiz geht.

Die n Privacy-Aktivisten der Gruppe SnowPen wollen ein Treffen organisieren, um das weitere Vorgehen zu diskutieren. Leider können sie lediglich über das Internet kommunizieren. Nachrichten werden asynchron übermittelt und sind nicht authentifiziert. Angenommen, die NSA kontrolliert $f = 1$ Verbindungen im kompletten Verbindungsgraphen zwischen den SnowPen Mitgliedern:

A) (5 Punkte) Können die SnowPen Mitglieder feststellen, dass Nachrichten verändert wurden, für $n > 1$?

B) (7 Punkte) Können sich die Mitglieder auf einen Treffpunkt und eine Zeit einigen, wenn eine Verbindung unter Kontrolle der NSA ist und es n SnowPen-Mitglieder gibt, für $n > 1$? Beweisen Sie Ihre Aussage.

C) (3 Punkte) Gibt es Fälle, in denen die NSA das Treffen nicht verhindern kann, obwohl die Anzahl kontrollierter Verbindungen $f \geq n/3$ ist? Begründen Sie Ihre Antwort.

11 Locking (15 Punkte)

Skiplisten (engl. *skip lists*) bauen auf einfach verketteten Listen auf und verbessern die durchschnittliche Laufzeit von Suchoperationen von $O(n)$ auf $O(\log n)$. Dazu befinden sich die Knoten ihrem Schlüssel nach sortiert in der Liste und besitzen zusätzliche Vorwärtsreferenzen.

Eine Skipliste mit k Levels enthält Knoten mit bis zu k Vorwärtsreferenzen. Die Knoten eines Levels $i > 1$ bilden jeweils eine einfach verkettete Liste, die eine Untermenge der Liste von Level $i - 1$ ist. Level 1 enthält schließlich alle Knoten.

Um nun einen bestimmten Schlüssel zu finden, beginnt man die Suche in der Liste des höchsten Levels, und steigt immer ein Level nach unten, wenn man den Schlüssel verpasst hat. Algorithm 1 (nächste Seite) zeigt eine Implementierung der Löschoperation.

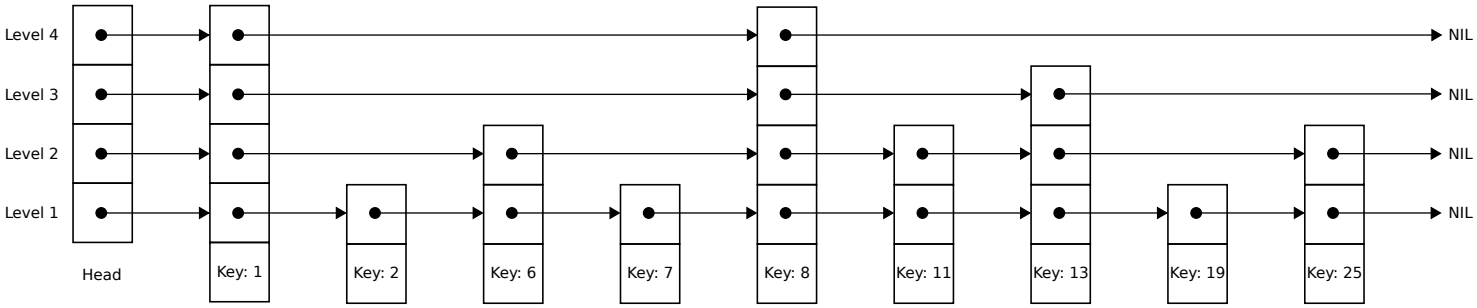

Abbildung 1: Beispiel einer Skipliste mit vier Levels

A) (3 Punkte) Ohne Locking können „böse Dinge“ passieren, wenn mehrere Threads gleichzeitig Operationen auf dieser Datenstruktur ausführen. Geben Sie ein Beispiel.

B) (6 Punkte) Ergänzen Sie das Grundgerüst von Algorithm 1 um Hand-over-Hand Locking. Verwenden Sie dazu *LOCK(n, i)* und *UNLOCK(n, i)*, die jeweils das Lock für Level i von Knoten n sperren/entsperren. Nicht jede Zeile wird benötigt. Es dürfen mehrere Statements pro Zeile verwendet werden.

C) (6 Punkte) Eine stark parallelisierte Anwendung verwendet Skiplisten, um Nachrichten von Produzenten- an Konsumenten-Threads zu übergeben. Viele Konsumenten lesen und entfernen einfach das Element mit dem kleinsten Schlüssel, was zu viel Contention an den Locks des Head-Knotens führt. Wie könnte man die Datenstruktur anpassen, so dass die Löschoperation ohne Locks auskommt? Welche Nachteile handelt man sich damit ein?

Algorithm 1 Element aus Skipliste mit k Levels entfernen

```
1:  procedure DELETE(key)
2:      n ← Head                                   ▷ Zuerst suchen wir nach dem Schlüssel
3:      .....
4:      for i ← k to 1 do
5:          .....
6:          while n.next[i] ≠ NIL and n.next[i].key < key do
7:              .....
8:              temp ← n.next[i]
9:              .....
10:             n ← temp
11:             .....
12:         end while
13:         previous[i] ← n                        ▷ Vorgänger für jedes Level merken
14:         .....
15:     end for
16:     .....
17:     n ← n.next[i]
18:     if n ≠ NIL and n.key = key then            ▷ Haben wir den Schlüssel gefunden?
19:         .....
20:         for i ← 1 to n.height do
21:             .....
22:             previous[i].next[i] ← n.next[i]    ▷ Umsetzen der Zeiger
23:             .....
24:         end for
25:         .....
26:     end if
27:     .....                                      ▷ Locks wieder freigeben (Aufgabenteil B)
28:     .....
29:     .....
30:     .....
31:     .....
32: end procedure
```

12 Game Theory (14 Punkte)

In der WG von Alice und Bob gibt es mal wieder das Problem, dass das Bad geputzt werden muss. Alice kann hoffen, dass Bob es übernimmmt, oder Alice kann es selber machen. Ein sauberes Bad ist Alice CHF 10 wert. Wenn Alice das Putzen erledigt, mindert dies ihren Nutzen um CHF 5. Bob ist ein sauberes Bad nur CHF 8 wert. Auch für Bob gilt, dass das Selber-Putzen seinen Nutzen um CHF 5 verringert. Sollte das Bad nicht geputzt werden, hat Alice einen Nutzen von CHF 0; Bob immer noch einen von CHF 2.

A) (6 Punkte) Modellieren Sie das Szenario als Spiel. Bestimmen Sie alle reinen Nash-Gleichgewichte, das Social Optimum und den Price of Anarchy.

B) (5 Punkte) Alice hat die Möglichkeit erhalten, eine Putzfrau für CHF 3 anzustellen (nur Alice, dieses Spezialangebot gilt nicht für Bob). Leider arbeitet die Putzfrau nicht so gründlich wie Alice oder Bob es tun würden, so dass sich der Wert eines sauberes Bad für Alice auf insgesamt CHF 6 reduziert (inkl. der Kosten für die Putzfrau). Bob merkt den Unterschied nicht. Allerdings mag Bob keine fremden Leute in der Wohnung, so dass er einen Nutzen von CHF 1 hat, wenn die Putzfrau ihre Arbeit erledigt und CHF 0, wenn Bob der Putzfrau beim Putzen auch noch hilft. Modellieren Sie dies als Spiel. Bestimmen Sie alle reinen Nash-Gleichgewichte, das Social Optimum und den Price of Anarchy.

C) (3 Punkte) Alices Mutter ist nicht begeistert von der Idee, dass sie sich eventuell eine Putzfrau leistet. Da die Mutter Alice und Bob sehr gut kennt, sieht sie, dass sie durch das Versprechen einer Zahlung von CHF 3 (an einen von beiden) bei genau einer Kombination verhindern kann, dass Alice die Putzfrau anstellt. Was hat Alices Mutter vor? Modellieren Sie dies als Spiel. Bestimmen Sie alle reinen Nash-Gleichgewichte, das Social Optimum und den Price of Anarchy.

13 Network Updates (8 Punkte)

Gegeben sei das Netzwerk in Abbildung 2 mit den vier eingetragenen Netzwerkflüssen. Die Netzwerkflüsse sollen zu der Konfiguration in Abbildung 3 geändert werden, jedoch sollen die änderungen konsistent sein, d.h. ...

A) (2 Punkte) Warum kann man nicht direkt alle änderungen zugleich ganz umsetzen?

B) (2 Punkte) Wenn man nur einen einzigen Netzwerkfluss ganz updaten möchte, welche(n) könnte man direkt updaten?

C) (4 Punkte) Geben Sie eine Folge von möglichst wenig Zwischenzuständen an, mit denen das Netzwerk zu der gewünschten Konfiguration geändert wird.

Die durchgezogenen Kanten haben eine Kapazität von 3 pro Richtung, die gepunkteten Kanten haben eine Kapazität von 5 pro Richtung. Es gilt $f_r = 2$, $f_b = 1$, $f_g = 2$ und $f_p = 3$.

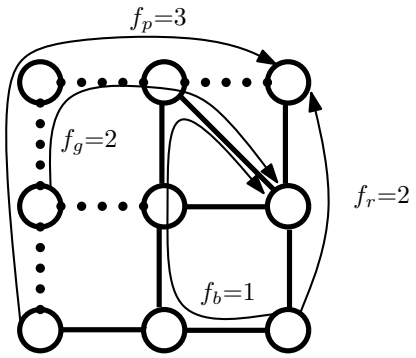

Abbildung 2: Ausgangskonfiguration.

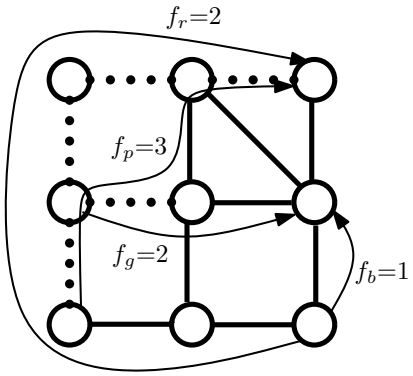

Abbildung 3: Zielkonfiguration.

14 Clock Synchronization (8 Punkte)

Sie arbeiten in einem Team an einem neuen GPS-Receiver. Die ersten Tests haben gezeigt, dass in vielen Situationen weniger als die benötigten vier Satelliten sichtbar sind und deswegen keine Position berechnet werden kann. Ihr Team diskutiert nun mögliche Lösungsansätze und entscheidet sich das GPS-System um folgende mögliche Sensoren zu erweitern:

- Uhr (exakt UTC synchronisiert, perfekte Genauigkeit)
- Höhenmesser (perfekte Messung)
- Beschleunigungsmesser
- Kompass

A) (3 Punkte) Wie könnten solche zusätzlichen Informationsquellen überhaupt eine Positionsberechnung mit nur zwei Satelliten ermöglichen?

B) (2 Punkte) Welche Sensoren aus der obigen Liste würden Sie verwenden, um das Problem zu lösen? (Kosten, Platz und Strombedarf vernachlässigen)

C) (1 Punkte) Einer Ihrer Kunden möchte nun Ihr GPS-System auf einen Kreuzfahrtschiff einbauen. Inwiefern liesse sich das System darauf optimieren?

D) (2 Punkte) Ein anderer Kunde möchte mit dem GPS-System seine Rechenzentren exakt zeitlich synchronisieren. Inwiefern liesse sich das bestehende System darauf optimieren?