ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed
Computing

HS 2012

Prof. Dr. Roger Wattenhofer, Dr. Thomas Locher
C. Decker, B. Keller, S. Welten

Prüfung
Verteilte Systeme
Teil 2

Freitag, 8. Februar 2013
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		20
10		14
11		20
12		20
13		16
Total		90

9 Multiple Choice (20 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) Konsensus

Aussage	wahr	falsch
Ein deterministischer Algorithmus kann in einem asynchronen System Konsensus garantieren, auch wenn ein Prozess abstürzt.	☐	☐
Ein f-resilient Konsensus Algorithm terminiert nach f Runden im Worst Case.	☐	☐
Jeder Algorithmus der wait-free ist, ist auch lock-free.	☐	☐
Die Ausfallwahrscheinlichkeit eines simplen Mehrheitsquorum-Systems geht gegen Null, wenn die Anzahl der Knoten gegen unendlich geht (Ausfallwahrsch. pro Knoten $p << \frac{1}{2}$).	☐	☐
Ein atomares Read/Write Register hat Konsensus-Nummer 0.	☐	☐

B) Consistency und Commit Protokolle

Aussage	wahr	falsch
Eine Client Partial Order definiert eine eindeutige zeitliche Reihenfolge aller Operationen eines Clients.	☐	☐
3PC funktioniert auch dann korrekt, wenn ein Client byzantinisch ist.	☐	☐
Paxos könnte anstelle einer einfachen Mehrheit auch andere Quorum-Systeme verwenden.	☐	☐
Eine einfache Mehrheit ist ein minimales Quorum-System (Minimal Quorum System).	☐	☐
Ein B-Grid hat eine asymptotisch kleinere Ausfallwahrscheinlichkeit als ein Grid.	☐	☐

C) P2P

Aussage	wahr	falsch
Jeder Lookup in einem P2P System mit n Peers benötigt mindestens $\log n$ Schritte.	☐	☐
MapReduce eignet sich gut für die Berechnung von Matrizenmultiplikationen.	☐	☐
Beim Consistent Hashing erhält man die IDs sowohl der Dokumente wie auch der Peers durch Hashing.	☐	☐
Man kann in einem P2P Netzwerk suchen, auch wenn man nur einen Knoten des Netzwerks kennt.	☐	☐
Butterfly und DeBrujin Graphen brauchen nur $O(\log n)$ Schritte pro Suche, obwohl die Knotengrade konstant sind.	☐	☐

D) Locking

Aussage	wahr	falsch
Mutual Exclusion bedeutet, dass immer genau ein Prozess in der Critical Section ist.	☐	☐
Bei Exponential Backoff Locks dürfen Prozesse, die den Lock noch nicht erhalten haben, abstürzen, ohne dass ein Deadlock die Folge ist.	☐	☐
Ein Test&Test&Set Lock ist schneller als ein Test&Set Lock, weil Test&Test&Set Locks mehrheitlich den Wert im eigenen Cache lesen.	☐	☐
Mit einem TAS kann man eine wait-free CAS Primitive implementieren.	☐	☐
Im Gegensatz zu MCS Locks testen CLH Locks einen Wert im eigenen Cache und sind deshalb schneller auf Non-Uniform Memory Architekturen.	☐	☐

10 Konsensus (14 Punkte)

In der Vorlesung haben wir gesehen, dass Compare & Swap (CAS) und Load Linked / Store Conditional (LL/SC) beide Konsensusnummer unendlich haben.
LL/SC ist für diese Aufgabe wie folgt definiert:

- int LL(memoryLocation): Liest den Ganzzahlwert an der angegebenen Speicheradresse und gibt ihn zurück.
- bool SC(int value): SC versucht den mitgegebenen Wert an die Stelle zu schreiben, von welcher der letzte LL Aufruf von diesem Thread gelesen hat. Sollte ein anderer Thread den Wert an dieser Speicheradresse verändert haben so schlägt SC fehl und gibt FALSE zurück, ansonsten (d.h. der Wert konnte geschrieben werden) TRUE. Falls LL zuvor noch nie aufgerufen wurde, wird auch FALSE zurückgegeben.

A) (8 Punkte) Schreiben Sie eine wait-free Implementierung (in Pseudocode) eines n-Konsensus-Algorithmus, die LL/SC und atomare Read/Write Register verwendet. Argumentieren sie kurz warum ihr Algorithmus wait-free ist.

B) (6 Punkte) Nun verändern wir die Implementierung von SC ein wenig: SC schlägt nun zusätzlich auch fehl, falls der Wert von einem anderen Thread mittels LL gelesen wurde. Adaptieren Sie Ihre wait-free Implementierung aus Teilaufgabe A) um auch unter diesen Umständen wait-freedom zu garantieren. Sollte nun kein wait-free Algorithmus mehr möglich sein, argumentieren Sie warum.

11 Quoren (20 Punkte)

Gegeben seien folgende zwei Quorum-Systeme:

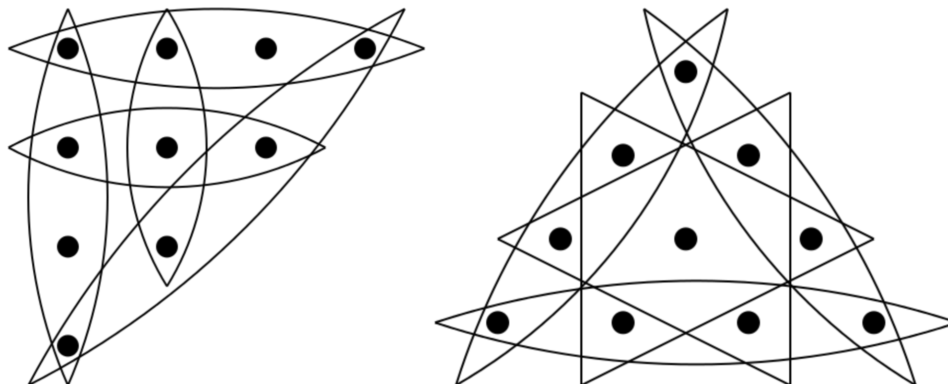

Abbildung 1: Links ist Quorum-System 1, rechts Quorum-System 2.

A) (2 Punkte) Warum ist das System 1 kein gültiges Quorum-System?

B) (8 Punkte) Bestimmen Sie die folgenden Eigenschaften des Quorum-Systems 2 (für eine uniform verteilte Zugriffsstrategie):

- Anzahl Quoren
- Work
- Load
- Resilience

C) (2 Punkte) Wie viele byzantinische Knoten kann das System 2 tolerieren?

D) (3 Punkte) Ist es möglich ein System zu kreieren, welches ebenfalls 10 Knoten wie System 2 hat, durch eine andere Quorenwahl aber eine höhere Resilience erreicht? Begründen Sie.

E) (5 Punkte) Wie gross ist die Anzahl Quoren und die Work für ein quadratisches B-Grid mit 36 Knoten und 3 Bändern?

12 Small Systems (20 Punkte)

Ziel dieser Aufgabe ist es einen Heap mit *Mutual Exclusion* zu programmieren. Ein Heap ist ein Binärbaum, bei dem der Wert des Parents kleiner ist als die seiner Children. Der Heap ist in einem Array gespeichert, so dass die Wurzel Index 1 hat und die Children von einem Knoten i jeweils $LEFT(i) = 2 \cdot i$ und $RIGHT(i) = 2 \cdot i + 1$ sind. Die Grundfunktionalität ist in Algorithm 1 und Algorithm 2 gegeben.

Algorithm 1 Wert einfügen

```
 1: procedure PUSH(A, VALUE)
 2:     i = 1
 3:     ....................
 4:     while A[i] != null do ........
 5:         next = smallestChild(i)
 6:         ..................
 7:         if if(A[i] > value) then
 8:             ................
 9:             exchange A[i] and value
10:             ................
11:         end if
12:         ..................
13:         i = next
14:         ..................
15:     end while
16:     A[i] = value
17:     ....................
18: end procedure
```

Algorithm 2 Kleinsten Wert entfernen

```
 1: procedure POP(A)
 2:     ....................
 3:     ret = A[1]
 4:     i=1
 5:     A[1] = ∞
 6:     ....................
 7:     while A[i] != null do
 8:         ..................
 9:         next = smallestChild(i)
10:         ..................
11:         exchange A[i] and A[next]
12:         ..................
13:         i = next
14:     end while
15:     ....................
16:     A[i] = null // Mark as not used
17:     ....................
18: return ret
19: end procedure
```

A) (4 Punkte) Wie würde man Coarse-Grained Locking implementieren? Was bedeutet das für gleichzeitige Zugriffe mehrerer Prozesse?

B) (8 Punkte) Ergänzen Sie das Grundgerüst von Algorithm 1 und Algorithm 2 um Hand-over-Hand Locking. Verwenden Sie dazu *LOCK(j)* und *UNLOCK(j)*, die jeweils ein Lock auf das $j$te Element im Array sperren/entsperren. Nicht jede Zeile wird benötigt. Es dürfen mehrere Statements pro Zeile verwendet werden.

C) (5 Punkte) Ist die Implementierung Deadlock-frei? Argumentieren Sie wieso kein Deadlock möglich ist oder beschreiben Sie ein Deadlock-Szenario.

D) (3 Punkte) Beim Hand-over-Hand Locking wird die Wurzel am Anfang von jeder Operation gelockt. Könnte man ein anderes Lockingverfahren verwenden um das zu verhindern?

13 Kurzfragen (16 Punkte)

A) (4 Punkte) Was ist die Konsensusnummer von *getAndIncrement*? Kann mit solchen Read-Modify-Write Registern, zusammen mit atomaren read-write Registern, ein *Shared Counter* implementiert werden? Skizzieren Sie eine Implementation oder argumentieren Sie, warum das nicht geht.

B) (3 Punkte) Gegeben sind zwei Knoten, maximal einer davon ist byzantinisch. Ist es möglich für diese zwei Knoten Konsensus zu erreichen unter der folgenden Gültigkeitsbedingung?
Der gewählte Wert muss ein Eingabewert von einem korrekten (d.h. nicht-byzantinischen) Knoten sein.
Zeigen Sie, wie Konsensus erreicht werden kann, oder beweisen Sie, dass es nicht möglich ist.

C) (3 Punkte) Wir möchten Konsensus lösen im synchronen Modell mit *fail-stop* Knotenausfällen. Gegeben sei die folgende Gültigkeitsbedingung: *Wenn alle Knoten mit dem gleichen Wert starten, dann muss dieser Wert gewählt werden.*
Betrachten Sie den folgenden Algorithmus:

1: Sende den eigenen Wert an alle anderen.
2: **if** alle erhaltenen Werte identisch **then**
3:     wähle diesen Wert
4: **else**
5:     wähle 0
6: **end if**

Zeigen Sie die Korrektheit des Algorithmus oder argumentieren Sie, warum der Algorithmus nicht funktioniert.

D) (3 Punkte) Welche Vorteile und Nachteile haben Queue Locks gegenüber TTAS Locks, TAS Locks und Backoff Locks?

E) (3 Punkte) Beim Lock-Free Fine-Grained HashSet werden alle Werte in eine einzige Liste eingefügt, allerdings in Split Order. Wozu dient diese Ordnung? Ordnen Sie die Zahlen von 0 bis 7 nach ihrer Split Order.