Distributed Data Management SS 2017
Prof. Dr.-Ing. Sebastian Michel
MSc. Kiril Panev
TU Kaiserslautern, FB Informatik – Lehrgebiet Informationssysteme
http://dbis.informatik.uni-kl.de
Exercise Sheet 6: Hand out 22.06.2017, Due 04.07.2017 - 16:00

Assignment 1: Causal Consistency (1 P.)

Given the following processes reading from and writing to a data store.

Process 1:	$R(x)x_0$		$W(x)x_1$		$W(z)z_2$					
Process 2:				$R(x)x_1$		$W(y)y_0$				
Process 3:				$W(y)y_1$	$R(x)x_1$		$W(x)x_2$			
Process 4:		$W(z)z_0$	$W(z)z_1$					$R(x)x_2$	$W(z)z_3$	

For clarification, consider the following example of a causal dependency graph. We denote a write to key k with unique version i as k_i, so the graph contains two writes: a write x_1 to key x and a second write y_1 to key y such that x_1 happens-before y_1:

$$x_1 \to y_1$$

According to the definition of causal consistency, if a process reads y_1, then its subsequent reads from x must return x_1 or another write x_i such that x_i is concurrent to x_1 or $x_1 \to x_i$. It is not causally consistent if a process reads y_1 then reads an older value x_0 with $x_0 \to x_1$ because happens-before is not respected.

(a) Draw the causal dependency graph between the individual data values x_i, y_i, and z_i and write next to each value the corresponding vector clock.

(b) Consider the following clients that read from the data store. For each of them, determine if, given the values read, the data store provides causal consistency or not. Explain your answers.

Client 1: $R(x)x_2$ $R(y)y_1$
Client 2: $R(x)x_1$ $R(y)y_0$ $R(x)x_2$
Client 3: $R(y)y_1$ $R(x)x_2$
Client 4: $R(z)z_3$ $R(y)y_0$
Client 5: $R(z)z_3$ $R(x)x_0$

Assignment 2: Consistent Hashing (1 P.)

Implement a simulation of consistent hashing to study the load balancing characteristics under different parameters. Create 100000 random numbers (or strings) as keys. Create 100 node identifiers. Use MD5 hashing and map the nodes to the id space of $0 \ldots 2^{32}$ by interpreting the MD5 hash of the node id as an integer (long), modulo 2^{32}. Assign the 100000 keys to the nodes according to the consistent hashing principle (smallest node hash that is larger than the key's hash; taking also care of the case around zero). Count for each node the number of distinct keys is obtained to store, i.e., its load (do not forget the zero counts). Submit next to the resulting plots and Gini values also the source code of your implementation.

(a) Compute the median, average, minimum and maximum load as well as the 25% and 75% percentiles of the load distribution (i.e., the load value that is larger than x% of the other load values) and create a plot (e.g., with Excel or Gnuplot) of the Lorenz curve of the load distribution. To do so, sort the load values in ascending order (on the x-axis). For each value on the x-axis, put the cumulative load on the y-axis. See `http://en.wikipedia.org/wiki/Lorenz_curve` for more details and explanations.

(b) Compute the Gini coefficient as an indicator for load balancing. (See `http://en.wikipedia.org/wiki/Gini_coefficient`). The Gini coefficient can be thought of as the ratio of the area that lies between the line of equality (perfect balancing, i.e., x% of the nodes have x% of the keys) and the Lorenz curve (of part (a)). Extend the implementation such that each node is having T virtual nodes, by placing the node T times in the ring. Compute for $T = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10$ the Gini coefficient as before.

Assignment 3: Potpourri (1 P.)

In order to mark this assignment as solved, you have to submit solutions to at least 15 *out of* 18 *questions.*

1. Assume there are n machines and m data objects stored in a consistent hashing scheme. What is the expected number of objects that are to be moved if an additional machine is added?
2. Sketch a computation in Spark that would benefit from rendering an RDD persistent.
3. Argue why the CAP theorem holds.
4. Name the three requirements for rendering a state machine t-fault tolerant.
5. Are fail-stop failures more difficult to handle compared to Byzantine failures, or the other way around, and why?
6. Enumerate and explain the three requirements of the agreement problem.
7. Why is agreement more difficult to reach, in general, if messages are sent in a unicast fashion, not in a multicast fashion?
8. Explain the difference between the concept of t-fault tolerance and Mean Time Between Failures (MTBF).
9. Given the design space of C, A, and P in the sense of the CAP theorem. Give one example system/application for each configuration of two-out-of-three properties with a brief explanation why the missing property would not fit in.
10. Assume the vast majority of nodes already know the latest version of an object. Is in this case the push- or the pull-based synchronization more efficient, and why?
11. Sketch an example showing that given two events x and y, from $C(x) < C(y)$ we cannot conclude that x happens before y, where $C(.)$ denotes the lamport timestamp of an event.
12. A client writes, in this order, the following values for a specific key: a, b, c, d, e, f, g, h. Then reads the value associated with that key and obtains values c and d. In which client-centric consistency models is this possible and why not in others?
13. Are Lamport timestamps assigned to events enough to render a state machine t-fault tolerant? If not, explain what else has to be done.
14. Argue why and why not weaker levels of database consistency are tolerable—give examples.
15. When are two vector clocks concurrent?
16. Given the setup in slide 36 of Lecture 7. What if $W < (N + 1)/2$?
17. Handling inconsistent versions of an object is difficult to achieve in general inside the data store. Give an example, different to the shopping basket example from the lecture, how inconsistencies can be handled in the application.
18. How could "read my writes" be implemented on top of an eventual consistent data store?