Download from the ACM Digital Library and read (you may skip Section 3):
Ajay D. Kshemkalyani, Ashfaq A. Khokhar, Min Shen: **Encoded Vector Clock: Using Primes to Characterize Causality in Distributed Systems.** ICDCN 2018: 12:1-12:8
https://dl.acm.org/citation.cfm?doid=3154273.3154305

Experiment Part 1

AIM
A goal of the project is to identify how fast the EVC grows, as a function of the number of events executed by a process, and the number of events executed by all the processes collectively. n is an input parameter. With n processes and assuming 32-bit integer, how many events it takes for the size of the EVC to occupy a number equal to 32n bits long. Once it equals 32n bits long, we can do a system-wide EVC reset. (Repeat the experiment assuming 64-bit integer instead of 32-bit.)

A POSSIBLE APPROACH
One way to measure this is to simulate asynchronous message-passing among n processes. Generate the first n prime numbers in Primes[1,n] and assign one to each process being simulated. Each process is simulated by a thread and the events of each process (internal, send, and receive events) are also simulated. A process generates internal and send events with a certain probability (a controllable parameter, which can also disallow internal events) and at a certain frequency (rate), say 1 event/ms. The events get queued in the process queue Q_i along with the simulation time timestamp, which is processed by the simulating thread. If it is a send event, its destination P_j is chosen at random from among the other n-1 processes. A corresponding receive event, (along with the sender's EVC timestamp) and along with the simulation time timestamp is enqueued in Q_j and processed (perform EVC operations for a receive event) by the thread simulating P_j. The simulation time timestamp of a receive event, as chosen by the sender, can be set to the sum of the send event simulation time timestamp plus a uniformly chosen value between, say, 0ms and 10ms.

The queue Q_i determines the schedule of events occurring at process P_i. The thread simulating process P_i dequeues events in simulation timestamp order, and simulates the EVC value update for that event. (That is, if an internal or send event, it does a multiply by the prime number associated with that process, if a receive event, it calculates the LCM, and then multiply, etc. etc. as per the EVC rules). In addition, the thread maintains a count of the number of simulated events it has processed after dequeueing from the local queue Q_i. For debugging purposes, it is useful to maintain 3 arrays: Send_simulated[1,n], Receive_simulated[1,n], and Internal_simulated[1,n] to count the corresponding number of send/receive/internal events dequeued and processed from each Q_i. In addition to the required EVC, it is useful to maintain the vector clock of the latest event simulated by each thread. (The sum of all the components of the vector clock at a thread gives the total number of events executed in the causal past in the system.)

A key challenge in the simulation is how to represent and store and manipulate (multiply, divide, and LCM or GCD calculation, etc.) EVCs that are hundreds or thousands of bits long. Check for example,
https://en.wikipedia.org/wiki/Arbitrary-precision_arithmetic
https://en.wikipedia.org/wiki/List_of_arbitrary-precision_arithmetic_software
You may, if you wish, choose to use one of the libraries for arbitrary-precision arithmetic (multiplication and division), and any appropriate programming language. For GCD calculation, you can implement Euclid's recursive algorithm.

DELIVERABLES

For selected values of n (e.g., 10, 40, 100), plot the size of EVC in bits as a function of the number of events executed in the system. Also, plot the (minimum) number of events executed per process, and the total number of events executed in the system, until the EVC size reaches 32n bits long, as a function of n. Repeat the experiments assuming size is 64n bits long. Note, you may want to use a logarithmic scale for the Y-axis. Also plot any other graphs showing relationships of interest as relates to the size of the EVC and the rate of its growth. For example, vary the above graphs by varying the mix percentage of internal events (baseline case is with no internal events) and communication events. You may plot not just graphs, but other forms of charts such as bar charts and histograms and pie charts, as you think are useful. If necessary, you may also tabulate data.

Analyze and explain the trends and observations you make about your data.

It is recommended that your plots be in Gnuplot. Document all the design choices you made in the project, and how you implemented the main procedures. Submit a detailed project report (hard-copy and soft copy), typeset preferably using Latex.

Note: The scientific approach to taking readings in simulations/experiments is as follows. For each setting of the parameters, take the readings for several runs (for example, 5 or 10 runs) and report the average. If there is noticeable variation in the readings (for the identical setting of the parameters), in addition to the average, report the standard deviation also for each setting of the parameters.

Experiment Part 2

AIM

To understand the benefits and limitations of storing and transmitting the logarithms of the EVCs (instead of the EVCs themselves). Due to finite-precision arithmetic, round-off errors may get introduced. Such errors may cause inaccuracies in the comparison test between the EVCs of two events. (Recall the comparison test, for events e and f, e → f if essentially EVC(f) mod EVC(e) = 0; now implement this using logarithms as shown in the paper, Section 4.4.)
An important part of understanding the limitations is being able to quantify the level of accuracy (or its complement, the error rate introduced) in the use of logarithms and anti-logarithms.

A POSSIBLE APPROACH

You will likely need to use a multi-precision binary floating point library.
Generate distributed executions for n = 10, 20, 40, say, containing up to, say (for example) v*n events totally. Probably v=50 is adequate, but you may have to iterate with this in the experiment. Store the full EVC (do not worry about the number of bits used, assume no bound) and/or alternatively, the vector clock, for each event. Also, for each event, store the logarithm of its EVC as generated by the algorithm of Figure 3. In order to store the logarithm, assume a bounded mantissa m (the digits after the decimal point). For example, assume m=64 bits, 128 bits, 256 bits, 384 bits, 512 bits, etc.

There are (v*n)(v*n – 1)/2 pairs of events for the total of v*n events. For each event pair, determine whether e -> f using the EVC or vector timestamp, and then using the logarithms of the EVC timestamps of e and f. A false negative for the causality relation is when e -> f but the use of logarithms does not infer so. A false positive for the causality relation is when e (NOT)-> f but the use of logarithms infers that e -> f. Calculate the percentage rate of errors (false positives and false negatives), for various n and m (after choosing a suitable value of v).

Examine the impact of the choice of the base on the results. For example, try b=2, 10, and some number comparable with the largest prime number used (say, the n^{th} prime).

Note, you may need to exercise some prudence in rounding off while taking the anti-log in the comparison test. For example, assume m=4. If the anti-log gives 12.1111, or 13.0001, you may want to consider it as 13 which is a natural number. It may be unlikely that the computer will give you 13 as the exact anti-log. This is because in any execution, the EVCs timestamps are not consecutive integers but rather, spaced far off, thus it may be reasonable to consider the anti-log as 13 (a natural number).

DELIVERABLES
Submit a hard-copy and soft-copy report quantifying the errors introduced by the rounding-offs in using finite-precision mantissas in the logarithms of the EVCs. Use graphs, and other forms of charts liberally. Decide what makes sense to present as results. For example, you may want to plot the percentage error rate (or degree of accuracy) of false negatives and false positives of the causality relation, for various n and m (and also try for different v and b). In addition, you can also use tables to present your data about the number and percentages of false positives and false negatives.

For the settings of n and m, you may have to experiment with the values, to see which ranges give meaningful results. (Also, what is a reasonable choice of v and b.) If you run out of system resources, you may need to revise the values chosen.

Analyze and explain the trends and observations you make about your data. Document all the design choices you made in the project, and how you implemented the main procedures. Make recommendations for enhancing the algorithm pseudo-code to further improve the accuracy. (For example, what choice b and m user should choose when using logarithms of EVCs. Or, when taking anti-logarithms, what extent of rounding-off to use.)