Causal Ordering Of Single Messages

Presentation By: Christian Newman

Original algorithm creators:
André Schiper, Jorge Eggli, Alain Sandoz

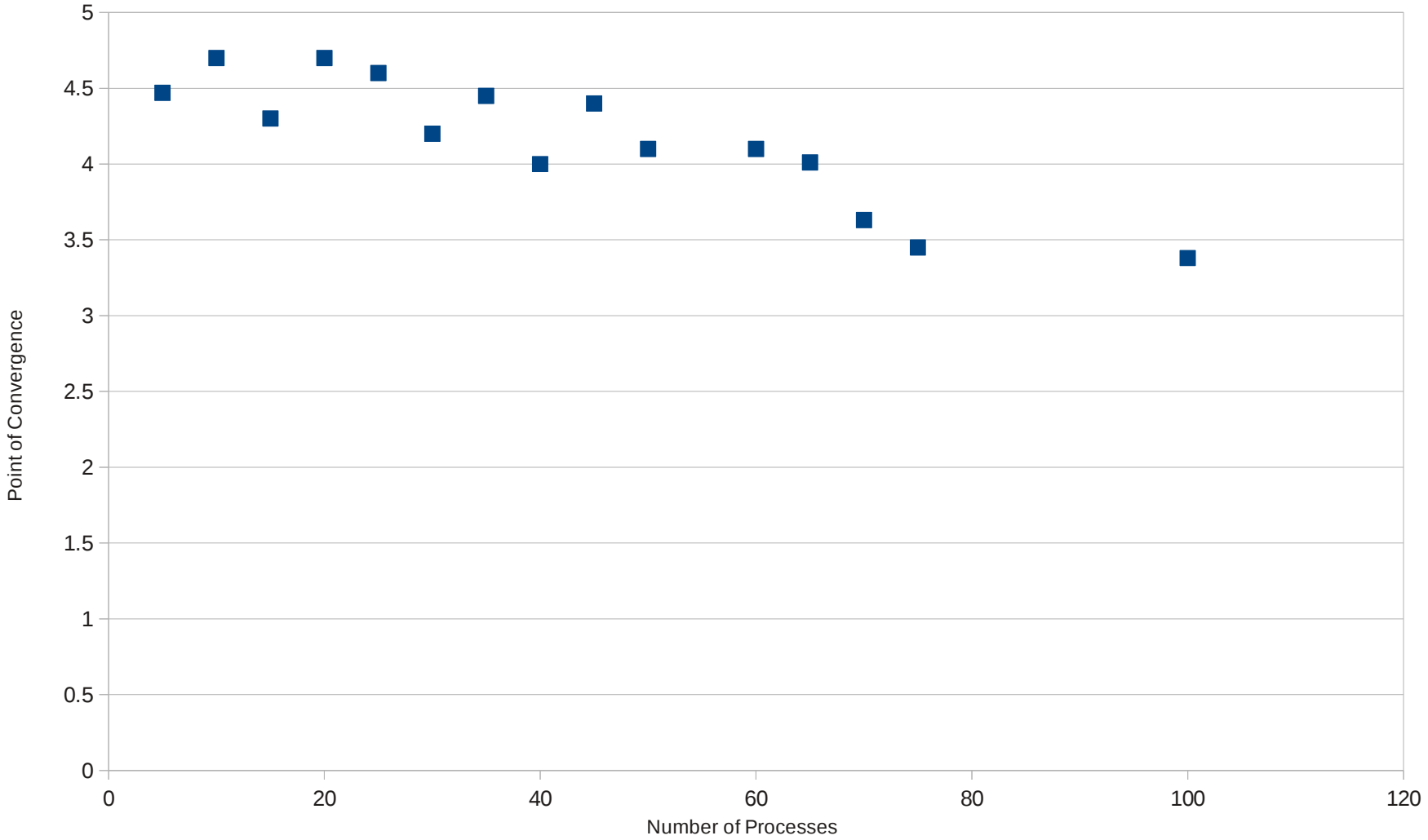
Algorithm Summary

- Store vector of last sent messages at each process and attach said vector to each message sent. When message is received, check vector to make sure that there are no messages still propagating that causally precede the current message.
- Update your own clock and vector of last-sent messages after making certain that the current message was received in order.

Setup

- Want to measure when the number of messages required to resolve message ordering conflicts converges to 1 message in two schemes:
- Randomly received messages (average case)
- Messages received in backwards order. (worst case)

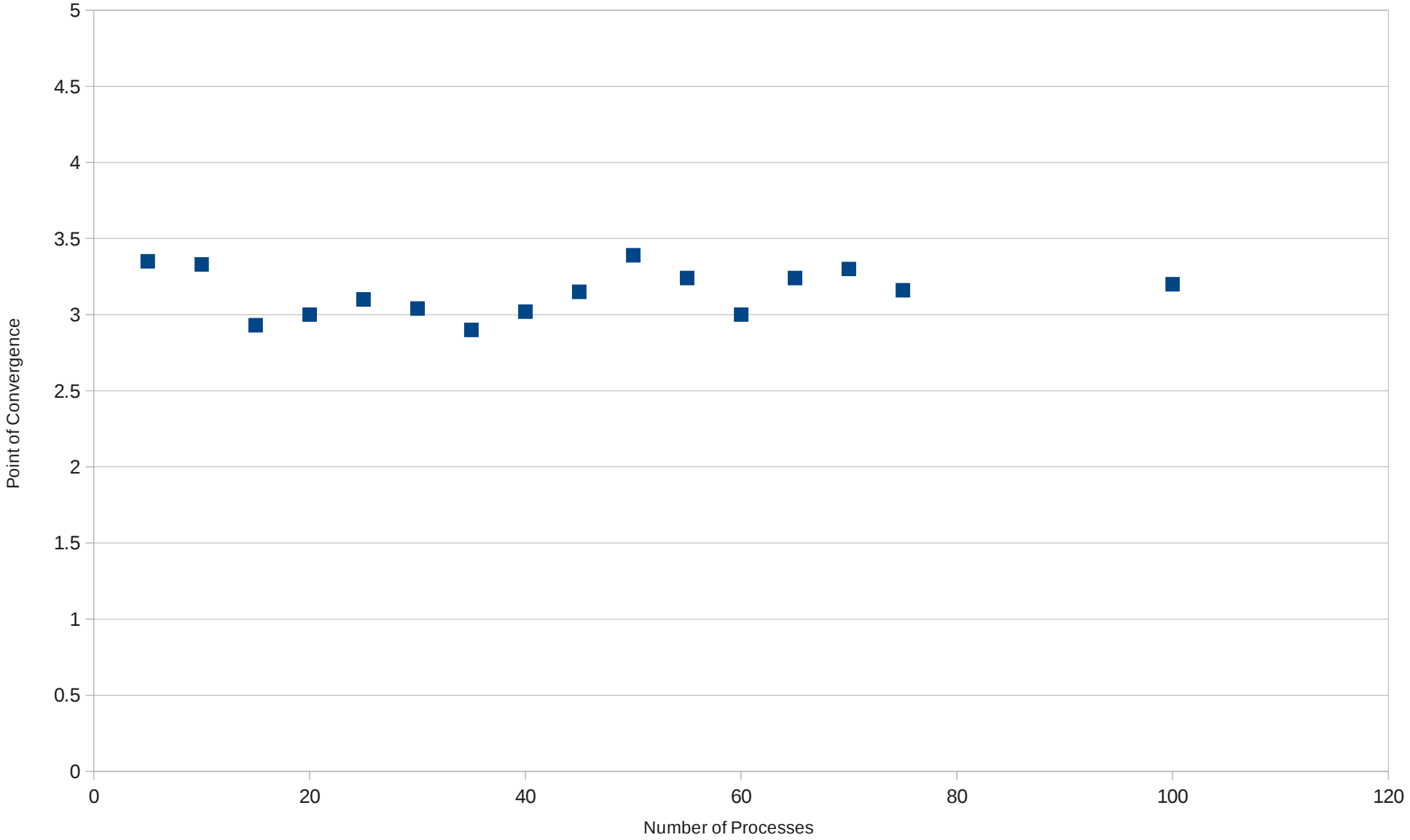
Results – Reverse-order receive



Results – Reverse-order Receive

- Range of convergence: 4.5 – 3.5 for 0 – 100 processes so, even in worst case, we get performance close average case (as we will see next)
- Slow tendency towards converging faster as number of processes increase.
- This is because, with higher number of processes, more messages are in the queue on average. When the message we were waiting for is finally received, many many messages are unbuffered.
- With higher number of processes, this happens to a greater magnitude.

Results – Random-order Receive



Results – Random-order Receive

- Range of Convergence: 3.5 – 2.9 but there is no conclusive dependence on number of processes.
- With Random-order the number of buffered messages pulled out as a consequence of causal relationship is lower than with Reverse-order.

Future Work

- Run tests with more processes, use more robust statistical metrics to help control the factor of randomness innate in the algorithm.
- Use an actual cluster to run tests.
- If building a simulation, use techniques to lower message size in order to run larger tests.

Code Defense

- Debugging: There're a lot of places for messages and the structures that help keep them in causal order to get jumbled.
- Solution: The output that the program used in the project 2 submission was created for the express purpose of debugging. Assert statements and catches for strange behavior made certain that assumptions were upheld.

Code Defense 2:

Code

End

- Questions?