# A Comparison Between Lamport's Scalar Clocks and Singhal & Kshemkalyani's Vector Clocks

Marling Engle
*Department of computer science*
*Kent State University*
*Kent Ohio 44242*
*mengle [at kent.edu]*

**Abstract**

*This paper looks at differences between Lamport's scalar clocks [2], and Singhal & Kshemkalyani's vector clocks [1]. However, because the two algorithms solve different types of problems the paper will focus on vector clocks and the cost of using them (when compared with using scalar clocks). We will be measuring the message length while varying the number of nodes in the network, and the total number of messages being sent in the network.*

## 1. Introduction

Both algorithms are logical clock algorithms – time is measured in terms of logical events (such as local events – events within the process, message send events, message receive events, etc). The clock value for a process P is represented as C(P). Next is a brief explanation of the algorithms.

### 1.1 Lamport's Scalar Clocks

Lamport's scalar clocks (herein known as scalar clocks) works by tagging each message with an integer, which is the clock value of the original process. Upon receipt of a message from B, the process compares the clock on the message (C(B)) to it's own clock (C(A)). If C(B) is greater than C(A) then C(B) is copied into C(A).

### 1.2 Vector Clocks

Vector clocks work on the same principle as vector clocks. However, instead of sending a single integer, the message is tagged with a vector. This vector is a collection of {id,C(id)} pairs.
Upon receipt of a message:

```
    my_clock=C(self)
    For each pair as {id,incoming_value}:
        C(id)=MAX(incoming_value,C(id)
        my_clock=MAX(my_clock,C(id))
    C(self)=my_clock
```

In this way, the local values are updated to the maximum of the local value and the carried value, and the processes own clock is always the maximum of everything it has seen.

### 1.3 Singhal & Kshemkalyani's Vector Clocks

Ordinary vector clocks require a vector containing a clock value for each process, to accompany every single message sent in the system. This happens even when it is not necessary (clock values are sent, which will not help the destination maintain its vector).

Singhal & Kshemkalyani's vector clocks (herein known as vector clocks) is a modification of ordinary vector clocks designed to reduce the overhead of traditional vector clocks. Their algorithm requires storing 3 vectors on each process, one is the original vector of vector clocks (the process's store values for other clocks). The other two vectors help the process decide which clock values to send to a given process. For example, when process A sends a process to B, it may only need to send {C(A),C(E),C(F)} (instead of all clocks from A-G). This set is specific to the destination. So if process A was to send that same message to process C, the vector tagged on the message may be different.

### 1.4 Motivation

These two algorithms accomplish similar tasks, but are not equal in their power. Vector clocks has functionality that supersedes that of scalar clocks. In fact, vector clocks can be used to emulate scalar clocks, by just tagging outgoing messages with a one clock long vector (the process's own clock). A functionality difference like this usually implies some sort of a cost, otherwise what is the point of having the scalar clocks. The following experiments are aimed at finding that cost.

## 2. Experiment – Base Algorithm

Clock algorithms such as these are made to tag messages. These original messages are part of a Base algorithm.

Base algorithm and network setup:

i. There are N nodes in the network.
ii. There is one initiator which sends T tokens to random nodes.
iii. Upon receipt of a token, the node randomly chooses a node from the N and sends to that node.

This base algorithm continually generates messages. I have decided on this algorithm because the rate at which these messages are generated can be varied by changing T.

### 2.1 Experiment – Setup

All experiments are carried out within a simulated environment. There is no message loss of any kind.

There will be two experiments. The first experiment is to measure the effect of the number of messages being sent, on the message length[1]. In this experiment, the number of nodes will be fixed at 11[2].

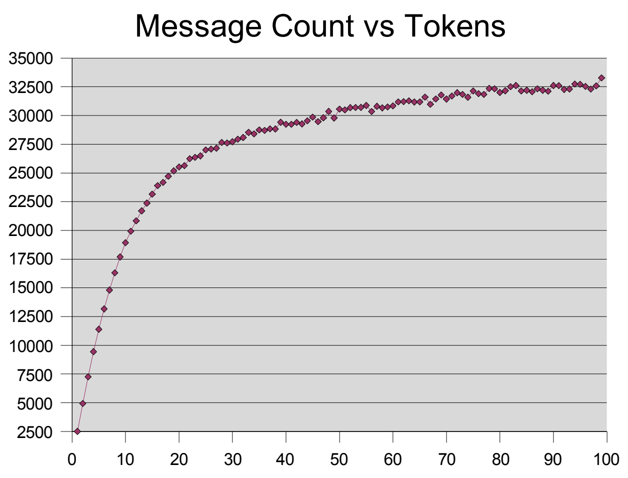
The second experiment is to measure the effect of varying nodes on the message length. For this experiment the number of tokens in the base algorithm is fixed at 3, simply to make sure that in even small networks (2 nodes) there will likely be a point in the computation where a node has no tokens.

Both experiments will terminate when any process reaches the logical clock value of 5000. 5000 logical clock ticks allows enough messages to be generated to measure the properties under investigation.

1 Message length is the number of clocks carried on the vector. All experiments are using average message length as the measure.

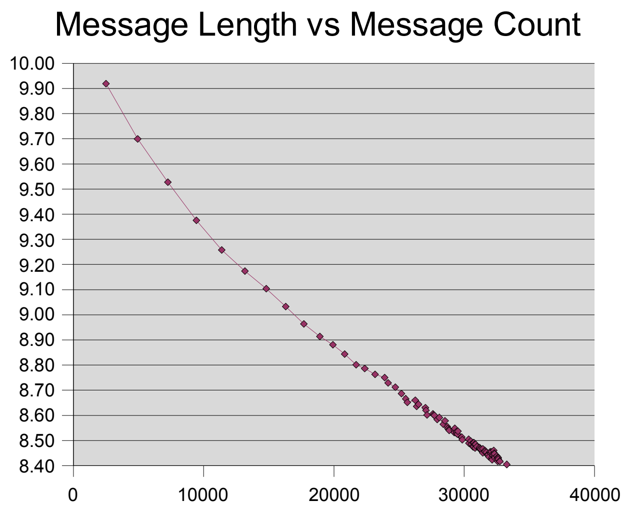
2 11 was the maximum number of clocks successfully carried because of TOSSIM's message length limitation.

### 2.2 Validation of token influence



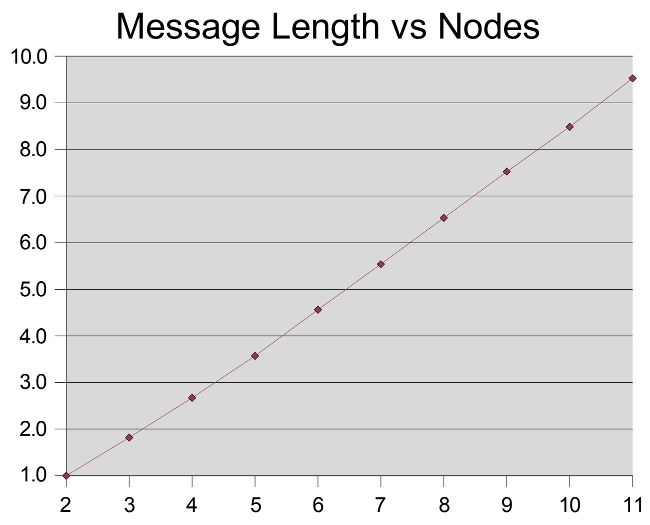
This figure is to validate that changing the number of tokens in the base algorithm directly influences the number of messages being sent. There is obviously a direct correlation. However it is interesting to note, that increasing the tokens yields less and less of an increase in message count. This is because the node count is fixed at 11, and after a certain number of tokens, it is most likely that all nodes always have tokens to send. Increasing the number of tokens past this point only marginally helps – it reduces the probability of a node ending up with no tokens at a given state in the computation.

### 2.3 Varying Message Count



Here we can see that increasing the message count decreases the message length. As in figure 2.2, there is less and less return for our input (increasing the message count in this case). We can also see that the most we ever save by using SK vectors (over traditional vectors) is 1.6 clocks per message. This is near the worse case scenario for SK vector clocks. The reason behind this, is that in a traditional base algorithm a message enters a subgroup of nodes, and causes the subgroup to perform some operations. In this type of algorithm, the locality of the nodes would give a much larger savings – each of the messages they send to each other, involves at most sending the clocks of their fellow subgroup members. However, in the base algorithm used for these experiments, there is no sense of locality, and so the message savings is not as great.

## 2.4 Varying the number of nodes



This experiment demonstrates that increasing the number of nodes is increasing the message length. The simple reason behind this is that there is more information to keep track of at each process, so there must be more information sent to them. It is also worth noting, that the rate at which this curve increases is determined by the message count (and the number of tokens, in the case of this experiment). An increased number of messages would decrease this curve's slope, because there would be a larger number of small messages instead of only a few longer messages.

## 2.5 Conclusions

The experiments above have demonstrated two main points:

I. Increasing message count will decrease the message length.
II. Increasing the number of nodes will increase the message length.

The cost of using vector clocks over scalar clocks is flatly defined by message length. Scalar clocks will always be a constant, but if the base algorithm requires strong consistency in it's logical clocks, than there may be no choice other than to implement vector clocks. This cost, however, can be minimized by reducing the number of nodes, increasing the number of messages between the nodes, or increasing the 'locality' effect discussed in section 2.3.

## 3. Future Work

While conducting these experiments, the influence of the base algorithm became increasingly apparent. The next step in this research, is to conduct similar experiments on base algorithms with different properties. Such algorithms could be wave algorithms, or tree build algorithms. Algorithms such as these would increase the 'locality' effect, and presumably enjoy a much greater savings in message length when using SK vector clocks.

Another issue left undiscovered is the reference to real time (as opposed to logical clock time). The experiments in this paper were terminated when a process reached logical clock value 5000. This happened much more rapidly in a case where more messages were sent (because each send/receive is a logical clock tick). Ending the experiment in terms of real time would allow for a greater variance in the number of messages being sent. This may change the perspective on the results because the cost of using vector clocks would also begin to factor in the speed at which processes are able to forward the tokens (of greater and greater size) along with the overhead of sending many small messages versus a single large message.

## 4. References

[1] M. Singhal, A. Kshemkalyani, “An efficient implementation of vector clocks,” *Information Processing Letters*, vol. 43, no. 1, pp. 47-52, August 1992.

[2] L. Lamport, “Time, Clocks, and the Ordering of Events in a Distributed System,”. *Communications of the ACM*, vol. 21, no. 7, July 1978.