# Akka Observations While Writing a Snapshot Store Plug-in

## Introduction

I finished writing a POC that performs better than the standard snapshot plugin that creates one file per snapshot per actor (a lot of files considering the tests were run on 150,000 actors). The POC is not a full implementation but it does demonstrate that there is a better way to store snapshots on a local disk than the standard implementation. It can save and retrieve a single snapshot per actor.

## Overall Results

### Home Rig

Testing on my home which is a base intel core i7 machine. The VM had 16GB RAM and 8 cores assigned. Snapshots were written to a local spinning disk, old 7200rpm and probably not the latest interface technology.

To send 150,000 msgs to 150K actors and to have each actor persist their state to disk, see actor state below, took on average 40 seconds and caused an MAX IO rate to the disk of 32Mbytes per second.

### Sandbox Rig

I am not sure of the CPU’s behind the sandbox machines but they are server technology and 4+ years old. The VM used had 20GB RAM, 16 cores (2 sockets, 8 cores each). This machine also achieved the same results of around 35-40s to send msgs and persist each of 150k actors.

## Actor State

Below are the objects properties that were assigned. The whole object was persisted and recovered by each of the 150K actors. Each actors state was randomly generated and stored in a flat file and loaded at startup.

public string AccountID { get; private set; }

public string CompanyIDCustomerID { get; set; }

public string AccountTypeID { get; set; }

public string PrimaryAccountCodeID { get; set; }

public int PortfolioID { get; set; }

public string ContractDate { get; set; }

public string DelinquencyHistory { get; set; }

public string LastPaymentAmount { get; set; }

public string LastPaymentDate { get; set; }

public string SetupDate { get; set; }

public string CouponNumber { get; set; }

public string AlternateAccountNumber { get; set; }

public string Desc1 { get; set; }

public string Desc2 { get; set; }

public string Desc3 { get; set; }

public string ConversionAccountID { get; set; }

public string SecurityQuestionsAnswered { get; set; }

public string LegalName { get; set; }

public string RandomText0 { get; set; }

public string RandomText1 { get; set; }

public string RandomText2 { get; set; }

public string RandomText3 { get; set; }

public string RandomText4 { get; set; }

public string RandomText5 { get; set; }

public string RandomText6 { get; set; }

public string RandomText7 { get; set; }

public string RandomText8 { get; set; }

public string RandomText9 { get; set; }

## Startup Observations

### Introduction

This section is focused on starting the code which would create the 150K actors and nothing else.

There is something I do not understand on starting up the 150K actors. It largely does not depend on whether the actors have a persisted state or not. The bulk of the time is going somewhere in the Akka framework. This needs a lot more investigation into what is going on. I tried a variety of configurations and was totally unsuccessful in changing the overall amount of time.

One worrying thing was that during the persist test, generally all threads were very active 80+% CPU busy, yet during startup generally some threads were not active at all (on configs with cores > 8) and when they were active they were only 30% CPU busy.

### Home Rig

Same setup as previous. I ran groups of 3 tests varying some parameter with largely zero impact. Generally starting 150k actors with no previous persisted state took 3mins and 39 seconds.

On the second & third runs, where the actors had persisted one copy & two copies of their states, the time was 3 minutes and 49 seconds. A whole 10 seconds for state recovery! During the subsequent runs where state was being read from the files and supplied to each actor, I could not reliably get disk read results, so they were so small as to be negligible! Again, reaffirming that the disk read has little to do with the recovery times.

### Sandbox Rig

Same setup as previous. It takes around 5 minutes to create 150K actors with no prior persisted state and 5 minutes 10 seconds with persisted state. Again, it seems that persisted state has a limited effect on the results.

## Variances tried

### Custom Dispatcher

I assigned a custom dispatcher to the actors, changing the number of threads allocated to the actor group. The default of 3 was used, then 2 and finally 10 threads. There was no marked difference on my home rig between each of the three settings. I confirmed the Dispatcher was set correctly by interrogating the actor in debug.

### Home Rig

For 10 thread count setting I got startup times of 3m:40s, 3:49, 3:51. For a thread count of 2 I got 3:39 and 3:51.

### Sandbox Rig

For 10 thread count the startup times were 4m:56 and for 2 thread count setting they were 4m:59s, 5:10