Tim Snyder
Dr. Cheng
NoSQL Data Mgmt
26 April 2018

Time Series & NoSQL

1. Introduction

Today, the need to store and analyze time series data is more relevant than ever for companies aiming to effectively optimize their services. From the constant collection of sensor data in autonomous vehicles to the tracking of complex interactions between millions of users online, the amount of data collected in modern systems creates a whole new frontier for developers to take advantage of. For example, the New York City Taxi Commission recorded 1.1 billion trips between 2009 and 2015 *(ref: "Case Study: New York City Taxis").* With this data, the commission was able to identify a pattern of unfair allocations of taxi services to pedestrian-heavy areas which resulted in an under-service of remote areas in New York's outer boroughs. For the development of these solutions, there is a need for flexible and scalable systems that accommodate the collection of these large datasets. Today, there is an abundance of systems available for utilization within the problem-domain presented by large datasets.

2. Time Series and our Model

Conceptually, time series are quite simple; they are no more than a series of data points indexed in time order. One example of a time series is the size of a grizzly bear population in yellowstone park recorded every month. Another example is the number of photons converted by a photodetector every 10ms. The scope of a time series dataset can range from hundreds of recordings every second to just 12 recordings over the course of an entire year. Time series datasets can be generated by observing the characteristics of a system or can be expressed with mathematical formulas. For our demonstration of NoSQL systems accomodation of time series data, we utilized a type of recurrence relation known as a logistic map to generate the dataset.

$$N_{n+1} = \alpha N_n \left(1 - \frac{(\alpha - 1)}{\alpha} \bullet \frac{N_n}{K_n}\right) ,$$
$$K_{n+1} = K_0 + \epsilon(-1)^n \quad , n \in Z^+ ,$$

Where N_n = size of population, K_n = carrying capacity, α = growth rate, ϵ = amplitude of oscillations of K
(ref: "Dynamics of a discrete population model with variable carrying capacity")

This particular recurrence relation seeks to model the size of a population over time but with the additional parameter of a varying carrying capacity which allows for a more accurate and dynamic modeling of a natural system.

3. Our Implementation

To implement this model, we utilized the Python runtime environment. While Python provides support for very expressive and dynamic programs, being designed as an interpreted language results in execution times that leave something to be desired when working with large volumes of data. For our application, we wanted the user to be able to query and analyze the dataset without having to worry about Python generating the dataset on the fly. Thus, some persistent storage solution was needed to handle the querying of our dataset after it was generated. For this, we utilized MongoDB, a document-oriented database with the PyMongo API serving as our interface. Our initial schema for the database was quite simple. It consisted of five fields with the primary key being an integer (0 → n) that identifies which generation the reading of the simulation was taken at. Typically, we took a reading of the simulation every generation.

```
{
        '_id':          ~Generation Number (0 → n)
        'pop':          ~Population Size
        'cap':          ~Carrying Capacity
        'amp':          ~Amplitude of Carrying Capacity Oscillations
        'alp':          ~Growth Rate
}
```

However, this schema did not adequately take advantage of the expressive nature of the document model provided by MongoDB. Each reading we took was inserted into the database as a unique document which is not very intuitive for time series data as many readings are often queried together by some user-defined range. To better support these range-based queries, we utilized MongoDB's ability to embed multiple readings within a single document, thus reducing the number of documents by a significant factor.

```
Document:
    {
        '_id' : ~Ranged index
        'val': [reading 0, reading 1, ... , reading N]
    }
A Reading:
    {
        'pop': ~Size of Population,
        'cap': ~Carrying Capacity,
        'amp': ~Carrying Capacity Modifier,
        'alp': ~Growth Rate
    }
```

With this new schema, it is possible to intuitively design documents in a fashion that easily facilitates different classes of query operations. For example, we can define a document whose primary key is one of the twelve months of the year. Then, each document contains an embedded list of key-value pairs which represent a day of the month accompanied by a reading of our model. Now, if a user wishes to query all readings from December, they have to query just one document. In our database, this document would be the 11th entry in a collection that holds one year's worth of documents. With the old schema, the user would have had to query 31 documents to access all of the readings from December.

4. Application of our Dataset

The analysis of time series data presents the opportunity to infer complex behaviors about a system with relative ease. For our presentation, we ran a few simulations of our model with each presenting a unique outcome that can be observed when the dataset is plotted on a graph. The first scenario resulted in the population entering a two-step growth and decline cycle after just a few generations. This trend continued indefinitely over time. *Note: the graphs below show time (in generations) plotted along the x-axis and population size plotted along the y-axis.*

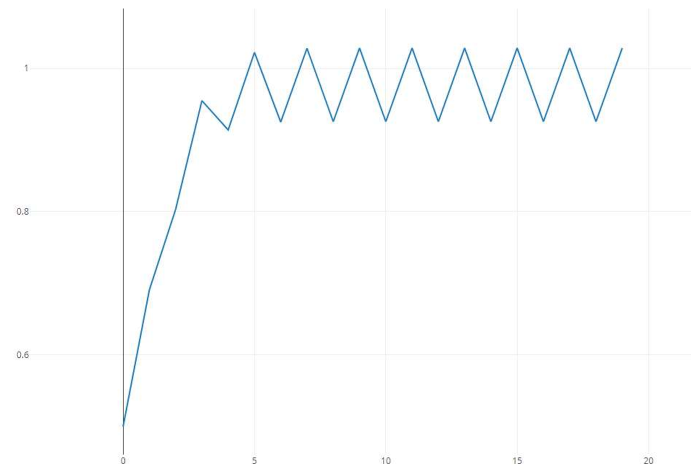

Figure A

Initial population:	0.5
Growth rate:	1.7
Initial capacity:	1.0
Oscillation amplitude:	0.1

Another scenario that was implemented resulted in a similar cycle as Figure A, but, these parameters resulted in a four-step cycle rather than a two-step cycle.

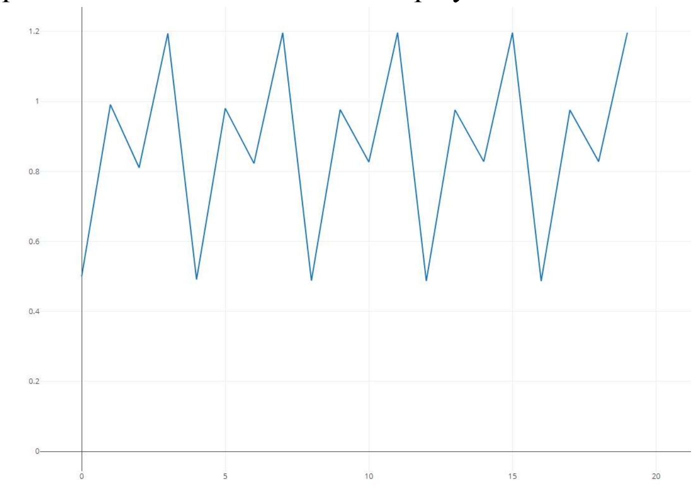

Figure B

Initial population:	0.5
Growth rate:	2.8
Initial capacity:	1.0
Oscillation amplitude:	0.1

Finally, the last scenario implemented resulted in the extinction of the simulated species. This was due to a carry capacity that increasingly oscillated to greater degrees until one generation found that the carry capacity had hit zero which represented an environment that cannot support a population.

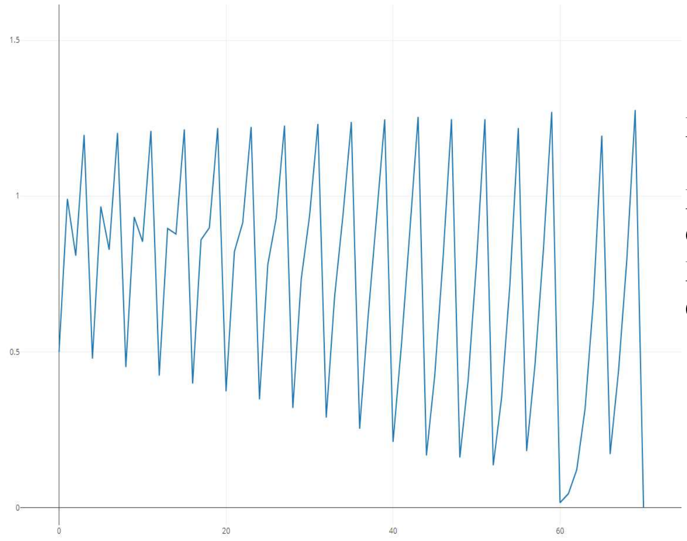

Figure C

Initial population:	0.5
Growth rate:	1.7
Initial capacity:	1.0
Oscillation amplitude:	$a_{n+1} = a_n * 1.01$

Figure A and Figure B scenarios referenced from "Dynamics of a discrete population model with variable carrying capacity"

<u>5. Conclusion</u>

While the data and scenarios presented in this implementation are not actually collected from real-world systems, it is still necessary to develop a system that effectively accesses and stores the data as if it is truly meaningful. Therefore, for our purposes, it was successful in playing its part in helping determine the path taken when developing the schema and programmatic query operations for our database. With the aid of MongoDB, we were able to almost instantaneously generate these graphs from our dataset. In the absence of MongoDB to facilitate quick access to our dataset, it would've taken a standalone implementation in Python many minutes to generate the dataset before it could perform any form of analysis. This sort of performance would not be as easily tolerated in a production environment where the datasets were larger and the analyses more complex. We found that even with basic queries, a simple data set, and trivial operations, a badly-designed schema results in poor performance that is easily avoided by considering the nature of the time series data that you're storing.

References

https://www.mssanz.org.au/modsim2015/A1/dose.pdf

https://en.wikipedia.org/wiki/Discrete_time_and_continuous_time

https://en.wikipedia.org/wiki/Logistic_map

https://en.wikipedia.org/wiki/Population_dynamics_of_fisheries

https://www.mongodb.com/presentations/mongodb-time-series-data

https://en.wikipedia.org/wiki/Recurrence_relation

https://www.equedia.com/how-fast-is-high-frequency-trading/

https://blog.timescale.com/what-the-heck-is-time-series-data-and-why-do-i-need-a-time-series-d
atabase-dcf3b1b18563

https://datasmart.ash.harvard.edu/news/article/case-study-new-york-city-taxis-596

Source code available at:
github.com/timothyCSnyder/discrete-population-model