# Automated Data Configuration for NoSQL Systems

*Kevin Lee, Akshay Mittal, Sachin Ravi*

## 1 Introduction

NoSQL storage options such as *Cassandra* and *MongoDB* are gaining traction in industry because traditional RDBMS struggle with the scale associated with web services. By being distributed and decentralized, these new storage services allow one to horizontally scale by adding more commodity machines as necessary. Though there exist many NoSQL options, all offering differing data models, we concentrate our efforts on column-oriented data stores, such as BigTable and Cassandra.

Data modeling in these new systems differs from traditional RDBMS data modeling. In relational databases, where most queries are carried out through the use of joins, a fully normalized data model is conventional; however in the NoSQL storage systems, we avoid using joins for scalability purposes and so it is necessary to denormalize one's data by storing repetitive information to optimize the system's response to certain queries.

The decision of how to denormalize one's data and to what level is dependent on the query workload one expects for his/her service. Not only does this require the developer to predict the workload ahead of time but it may involve a complicated cost-benefit analysis that is harder for larger, more complicated workloads.

Here, we believe it is possible to help the developer and have an automated way to do this modeling. Given a query workload and the normalized definition of how tables are related, it should be possible to give an optimal (or close to optimal) configuration for how the denormalized data should be stored. This sort of tool would help avoid extended analysis and allow one to automatically change the data configuration if the query workload has changed enough.

## 2 Related Work

We look to previous database literature on automatically enumerating materialized views for SQL databases to solve the above problem. In many ways, a materialized view is similar to a table that is storing denormalized data since in both cases, data is being replicated so that queries can be answered more efficiently. Specifically, in [1], the authors discuss an entire end-to-end solution for automated selection of materialized views. The main contributions of this paper are algorithms for: (1) producing a candidate materialized view set that is much smaller than the set of all possible materialized views for a query workload; (2) adding to this set by merging views from the original set.

In [1], however, the query optimizer is used extensively to evaluate the cost of materialized views for different queries. In Cassandra, for example, we do not have access to a query optimizer so it will be necessary to build our own cost model to evaluate the cost of a certain data configuration for a specific query.

Nectar [4] system was designed to avoid manual management of data and computation; it automates and unifies the management of data and computation within a datacenter. Derived datasets, which are the results of computations, are uniquely identified by the programs that produce them, and together with their programs, are automatically managed by a datacenter wide caching service. Any derived dataset can be transparently regenerated by re-executing its program, and any computation can be transparently avoided by using previously cached results. This trade-off between storage and computation can be leveraged to define an efficient view materialization scheme for NoSQL storage systems.

HBase [3] is an open source, non-relational, distributed database modeled after Google's BigTable. To make best use of the features of HBase, application developers have the onus to denormalize the schemas. If the query load changes, the initial optimal denormalization may prove to be inefficient for the incoming queries. An automated denormalization of the schemas will incur initial processing cost but will relieve the developer of the denormalization task.

## 3 Preliminary Plan

We plan on working with Cassandra, an open-source, column-oriented data-store created at Facebook, to evaluate our automated data configuration system. In the first iteration, we plan to work outside the system and create a prototype that accepts a list of Cassandra queries and a normalized view of the data, to produce an optimal denormalized data configuration. Building this system outside of Cassandra will allow us to concentrate on the actual project rather than spend a lot of time familiarizing ourselves with Cassandra code. We can then evaluate the automated data configuration produced against other manually planned options for various query workloads to see how well our system performs.

For a later goal, we would like to incorporate our system into Cassandra itself so that it can read from the query logs and automatically shift the existing data configuration to a more optimal one. We plan to use the "Yahoo! Cloud Serving Benchmark" (YCSB) framework [2] for the purpose of query workloads.

## References


[1] Sanjay Agrawal, Surajit Chaudhuri, and Vivek R Narasayya. Automated selection of materialized views and indexes in sql databases. In *VLDB*, volume 2000, pages 496–505, 2000.

[2] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 143–154, New York, NY, USA, 2010. ACM.

[3] Lars George. *HBase: the definitive guide*. O'Reilly Media, Inc., 2011.

[4] Pradeep Kumar Gunda, Lenin Ravindranath, Chandramohan A Thekkath, Yuan Yu, and Li Zhuang. Nectar: Automatic management of data and computation in datacenters. In *OSDI*, pages 75–88, 2010.