Aeneas: a tool to enable applications to effectively use non-relational databases

Cesare Cugnasco, Roger Hernandez
Barcelona Supercomputing Center
Barcelona, Spain
Email:{cesare.cugnasco, roger.hernandez}@bsc.es

Yolanda Becerra, Jordi Torres, Eduard Ayguadé
Universitat Politècnica de Catalunya - Barcelona Tech.
Barcelona supercomputing Center
Barcelona, Spain
{yolandab, torres, eduard}@ac.upc.edu

***Abstract*—Non-relational databases arise as a solution to solve the scalability problems of relational databases when dealing with big data applications. However, they are highly configurable prone to user decisions that can heavily affect their performance. In order to maximize the performance, different data models and queries should be analyzed to choose the best fit. This may involve a wide range of tests and may result in productivity issues. We present Aeneas, a tool to support the design of data management code for applications using non-relational databases. Aeneas provides an easy and fast methodology to support the decision about how to organize and retrieve data in order to improve the performance.**

***Keywords*—Non-relational database; distributed data-store; data model design; benchmarking; Apache Cassandra**

I. Introduction

The overwhelming amount of data that is currently invading all social fields has made big data applications become a hot topic in the research area [1]. One of the main challenges to face is how to store and organize all information in order to offer efficient and reliable access to users, and to make it easy to share all information among various users with different interests and requirements, and many times, those requirements come together.

Non-relational databases arise as an alternative to traditional databases to deal with the challenges big data applications pose[2], [3], [4], [5]. Advances in this area, together with the improvements in cloud computing technologies are opening new options in varied application fields to exploit the new power to store huge amounts of data.

However, the performance of non-relational databases is very sensitive to user decisions[6] as, for example, data organization, type of query, and configuration parameters such as data partitioning, number of replicas, amount of memory allocated, etc. This means that, in order to get the best possible performance, the user should test a lot of different organizations and parameters. This requires the implementation of various versions of data generator applications and data consumer ones, as this code depends on the data model, and a framework for measuring their performances. Moreover, the behavior of each configuration depends not only on the particular cluster but also on the particular workload that is being tested. Thus, the experimentation has to be repeated each time that the amount of involved data stored or the target data for a query changes.

In this paper we present Aeneas[1], a highly configurable tool to support the design of data management code for applications using non-relational databases. The user can configure this tool by providing the data model, the query parameters, and the performance metrics that he wants to evaluate. Then, Aeneas automatically generates the code to define the specified data model, to insert and to retrieve the data, and to evaluate the performance of accessing these data. Thus, Aeneas provides an easy and fast methodology to support the decision on how to organize and retrieve data with good performance, whilst solving the productivity limitations that could arise when using non-relational databases.

The rest of the paper is organized as follows: section II describes the database used by the current implementation of Aeneas and section III presents the design of the tool and the main aspects of its implementation; section IV describes a case study based on an application from the life sciences domain and section V shows the conclusions.

II. Background

Apache Cassandra[3] is a distributed database management initially developed by Facebook for internal usage, and later released as an open source project. It inherits the data model structure from Google BigTable[2] and the data replication and distribution management from Amazon's Dynamo[4]. We have chosen this database because it is highly configurable, making it possible to adapt it to the requirements of existing applications.

Data is organized in *column family* objects, which are similar to tables in the relational database model, and each column family contains a set of *columns*, which are comparable to attributes. A set of related columns compose a *row*, which is identified by a key. The row key is the main resource used to locate the data, and it is also used to distribute the data across the cluster nodes.

Cassandra does not support relationships between column families, such as foreign keys and join operations. Knowing this, the best practice when designing a data model is to keep related data in the same column family, duplicating the data if required.

The architecture of Cassandra is completely decentralized. All the nodes in a Cassandra cluster are equivalent: they all can

[1]Aeneas was one of the Cassandra relative's that, after Cassandra prophecy was intended to be Rome founder

receive read/write requests and are able to forward in parallel those requests to the hosting nodes.

III. Aeneas

The main goal of Aeneas is to offer a methodology to increase the productivity of the evaluation process, which will lead the user to get the best configuration parameters to access data. In this section, firstly we describe the main steps involved in the evaluation of each of these configurations and how to develop them without using Aeneas. Secondly, we describe the main aspects of the Aeneas design and implementation.

A. Evaluation Methodology without using Aeneas

Let us consider that we have different data models and different parameters to test. Then the main steps involved in the evaluation are the following.

Configure the cluster according to the parameters: In terms of database configuration, in a regular environment the user would be required to configure all of the cluster and database parameters such as replication factors and strategies, cache configurations, partitioning strategies, etc. Moreover, it is necessary to specify how the data is going to be organized in the database. To this end, the user should use the proper interface function and should specify all the organization characteristics.

Load all the data according to the data model to test: In order to populate the database the user should write the specific code to insert the data according to each model.

Generate queries workload: Once again, the code to query the database depends on the data model configuration. This means that for each configuration to test, the user should implement a version of the query code to access the database. In addition, in order to evaluate the concurrent access of several clients, the user should provide an extra code to control the execution of all the applications instances.

Collect statistics about the database system behavior: Up to this point everything has been prepared, but it is necessary to define the metrics and to implement the tools to collect statistics about the performance of the system.

Summarizing, all the actions mentioned above would have to be brought into consideration, executed and repeated for every experiment that has to be tested, and it should be made within the environment of the original application. Aeneas implements an evaluation environment that provides the user with an easy way to describe different data models, to specify different performance metrics and to perform and evaluate different data queries.

B. Aeneas design and implementation

Aeneas is composed by three main modules: Loader, Workloader and Analysis reporter. Thus, each user can use just the module that he needs to support his requirements. Moreover, as it is based on interfaces, any specific case study could be implemented by just extending those patterns and implementing the required services. Even when adding those new capabilities, the user interface and the module integration will not change. In addition to this, the abstraction layer can greatly improve the performance of data loading stage by managing and optimizing the insertion operations.The current implementation of Aeneas only supports the Apache Cassandra database. However, notice that the modularity of Aeneas makes it easy to add different modules to support different databases. Figure 1 represents the main steps in the evaluation process and the main components of Aeneas.

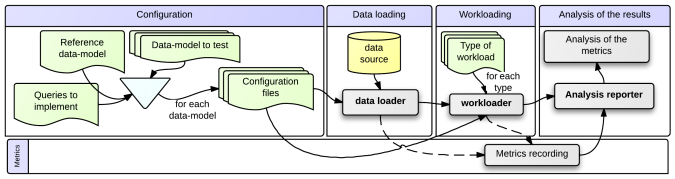

Fig. 1. Steps in the evaluation of a data model using Aeneas

The *Loader* is in charge of the configuration of the system and the loading of the input data. The main complexity involved in this task is to obtain a code independent of any data sources and of any possible data models. To achieve this, we have decoupled the definition of the data structure from the implementation of the access code. On one side, the user describes the structure of the data to be read, defining a grammar through a meta language. On the other side, we have defined a generic interface that allows to implement a specific reader for different data sources. Each loader uses the grammar to read information and to store it using the proper structure. Then, Aeneas reorganizes and transforms the read data in order to fit them into a specific model.

The *Workloader* performs a set of queries following different statistical distributions. The input for this module is the following: which of the implementations is to be tested, the number of tests to perform on it and the type of distribution used to generate the query input arguments.

The *Analysis reporter* retrieves metrics from both the client and the database sides, considering all the nodes in the cluster, and stores them in a consistent data store. By default, Aeneas gathers the most typical metrics such as the time required to populate the cluster, the statistical behavior of a query response time, memory usage, I/O and network frequency, and additional internal metrics from Cassandra as, for example, the amount of pending tasks per node but it is easily possible to add new metrics.

IV. Case study

Life science applications constitute one of the fields that can benefit from using non-relational databases. A usual scenario in this domain is to have a long process generating a huge amount of information that can feed other applications. Some of these analysis may be required long after the data generation, with different purposes and using different subsets of the generated data. Thus, it is convenient to keep this data stored as long as possible, to avoid wasting time to generate it again. In fact, life science researchers aim to have shared pools with such information (RSCB protein data bank[7] and the molecular dynamic extended library[8] are two examples of this kind of effort). In this domain, our case study is focused on applications working on molecular dynamics, which is a technique based on computer simulation that represents the evolution in time of complex systems, modeled at atom level [9]. The output of the simulation is the trajectory of a molecule, that is, the coordinates of each atom in time. An application analyzing these trajectories may need information about all the atoms during a particular range of time; or it may need information about a subset of atoms throughout the whole simulation.

Given these types of queries, we can consider two different data models: in the first one, each row represents the coordinates that a given atom has in each snapshot of the trajectory, and the row key is an atom identifier; in the second one, each row represents the coordinates of all the atoms in a given snapshot of the trajectory, and the row key is the snapshot identifier. In a generic dataset, the number of atoms is several order of magnitude smaller than the number of samples. This produces data models which mainly grow on one dimension. Since Cassandra distributes the data along the cluster nodes basing on the key value, and persist the data ordering by rows, there is a general trade-off between parallelism and efficient I/O devices usage.

We have used Aeneas to perform hundreds of experiments in order to get the best configuration to request the data for one trajectory. To do this, we only needed to adapt the configuration files according to each test. For example, one set of experiments has been devoted to deciding how to split a query, which is necessary to fit each sub-query results to the amount of memory available. We have analyzed how changing the amount of rows requested per each sub-query, and consequentially the number of sub-queries performed, may affect the response time of the accesses. We have observed that the performance of the accesses increases as the number of sub-queries decreases until reaching an inflection point. This inflection point depends on several factors: the amount of rows involved per sub-query, the amount of data requested per sub-query, the resources available in the cluster and the amount of nodes in the cluster. Thus, given a cluster configuration, in order to get the best performance it is necessary to use a query organization and a data model that allows to get a parallelism degree and a memory usage close to the values that characterize the inflection point. We have found that depending on the query organization it is possible to improve the performance of one of the queries by 13 times.

V. Conclusions and Future work

We have presented Aeneas, a tool which implements an analysis environment that provides the user with an easy way to describe different data models, to identify different metrics and to execute and evaluate various queries. Its usage dramatically increases the productivity when designing and tuning a non-relational database.

As part of our future work, we plan to add new features to Aeneas, such as support for automatic generation for a wider range of query types. Additionally, we plan to provide Aeneas with a model that is able to suggest query plans to maximize the memory usage while reducing node congestion.

Acknowledgements

This work is partially supported by the BSC-CNS Severo Ochoa program and the TIN2012-34557 project, with funding from the Spanish Ministry of Economy and Competitivity and the European Union's FEDER founds.

References

[1] R. Kouzes, G. Anderson, S. Elbert, I. Gorton, and D. Gracio, "The changing paradigm of data-intensive computing," *Computer*, vol. 42, no. 1, pp. 26 –34, jan. 2009.

[2] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 4:1–4:26, Jun. 2008.

[3] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010.

[4] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 205–220, Oct. 2007.

[5] L. George, *HBase: The Definitive Guide*, ser. Definitive Guide Series. O'Reilly Media, 2011.

[6] E. Hewitt, *Cassandra: the definitive guide*, ser. Definitive Guide Series. O'Reilly Media, 2011.

[7] "Rcsb protein data bank." [Online]. Available: http://www.rcsb.org/pdb/home/home.do

[8] "Model: molecular dynamics extended library." [Online]. Available: http://mmb.pcb.ub.es/MoDEL/

[9] R. Petrenko and J. Meller, *Molecular Dynamics*. John Wiley Sons, Ltd, 2010.