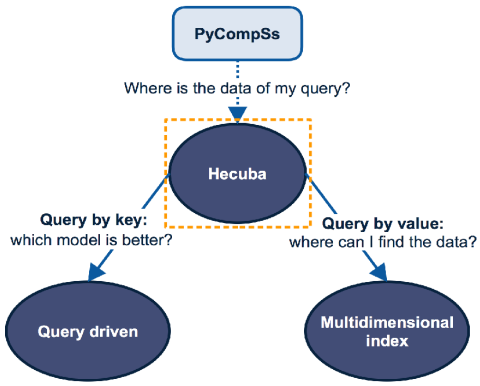
Hecuba: NoSql made easy

Guillem Alomar, Yolanda Becerra, Jordi Torres
Barcelona Supercomputing Center
guillem.alomar@bsc.es

Abstract*—*Non-relational databases are nowadays a common solution when dealing with huge data set and massive query work load. These systems have been redesigned from scratch in order to achieve scalability and availability at the cost of providing only a reduce set of low-level functionality, thus forcing the client application to take care of complex logics. As a solution, our research group developed Hecuba, a set of tools and interfaces, which aims to facilitate programmers with an efficient and easy interaction with non-relational technologies.

Keywords— Cassandra, PyCompSs, HPC.



I. Introduction

Hecuba is a set of tools that facilitates programmers anefficient and intuitive interaction with non-relational databases. We have added to Hecuba the implementation of the interface necessary to provide PyCOMPSs with a Storage Backend suitable to support Big Data applications, which in our case currently is Cassandra.

Cassandra implements a non-centralized architecture, based on peer-to-peer communication, in which all nodes of the cluster are able to receive and serve queries. Each node of the cluster is assigned a token. A partitioner function uses this token to decide how data is distributed among the nodes in the shape of a ring. Data in Cassandra is stored on tables by rows, which are identified by a key chosen by the user. Cassandra stores data by rows, and one node is responsible for hosting a specific row. The target node is chosen based on the key of the row and of the token of each node by the partitioner algorithm. In order to guarantee data availability, Cassandra can be configured to keep several replicas for all data.

The mapping of a Python dictionary on a data model in Cassandra is straightforward as both consist on values indexed by keys.We have decided to map each class containing one or more Persistent Dictionary on a Cassandra table. How to implement a class backed up by Hecuba: It is just necessary to indicate that the class is a subclass of StorageObj, which is a class implemented inside Hecuba, and that contains all the methods that are necessary to access and manipulate data backed up by Cassandra. Thus, the code of the application is mostly independent of the data backend used. Notice that programmers can add their own methods to the class definition.

II. Interface Implementation

Following we describe the implementation of the interface implemented by Hecuba as part of the StorageObj class. The methods exported to programmers allow to perform the following tasks:

- Make an object persistent: the implementation of this method creates the Cassandra table that it is necessary to hold the object and populates it with the data that the object had in memory.
- Object instantiation: if the constructor receives a parameter, the constructor binds the instantiated object with the corresponding Cassandra table, translating all the following accesses to that object into accesses to that table. On the contrary, if the constructor does not receive a parameter, all data associated to that new object instance will be kept in memory until the makePersistent method is executed.
- Query/Update data: if the object is in memory, the implementations of these methods are translated into the usual methods of a Python dictionary; if the object is backed by a Cassandra table, then the implementations of these methods consist on queries performed on the Cassandra table.
- Data iteration: the implementation of the keys method of a Persistent Dictionary returns a block of keys that will be the input parameter of a task. In order to enhance data locality, we decided to create the blocks of keys based on their node location. We have implemented two diferent iterators: one of them to create the diferent blocks of keys and the other one to traverse all the keys in a block (Figure 1).

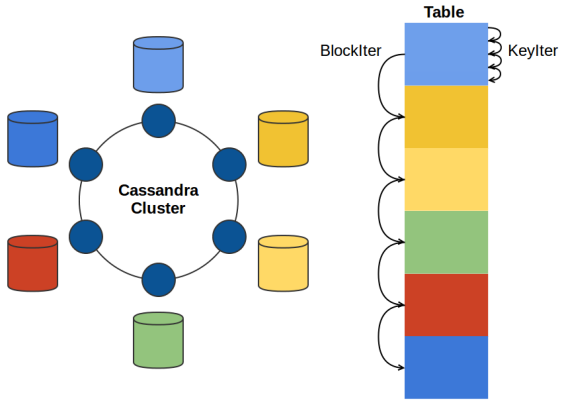


Figure 1: Representation of Hecuba Iterators.

- Delete a persistent object: this operation is implemented by the delete persistent method of the StorageObj and it just deletes from the database the table that was backing the object.

III. Data Locality

PyCompSs not only enables an easy way to define which functions will be parallelized, but also allows Hecuba to exploit data locality.

PyCompSs has at its disposal the location of all blocks of data that Hecuba has stored in Cassandra. Whenever a task needs to be run in a node, PyCompSs will decide to which node this task will be sent, depending on the location of the stored data that the task needs.

PyCompSs has a score function, which gives a higher priority to nodes with the most information needed by the task, and in case that this first node is not available for any reason, the task will be sent to the following node with higher priority. As data in Cassandra is replicated in many nodes, a task that needs more than one block of data will be sent to the node with the most needed blocks available (Figure 2).

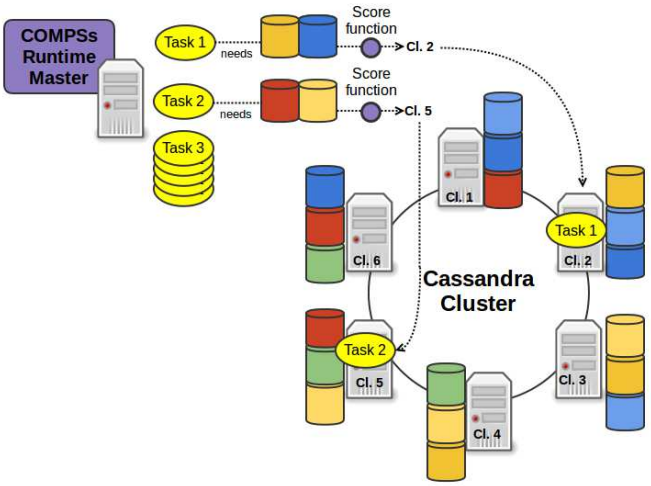


Figure 2: Representation of how COMPSs exploits data locality in an examples with tasks needing 2 blocks of data in a Cassandra Cluster with data replicated in 3 nodes.

Acknowledgments

The research leading to these results has received the support of the grant SEV-2011-00067 of Severo Ochoa Program, awarded by the Spanish Government.