UniDQP: Making MapReduce Scheduling Decisions with an Awareness of the Cached Data Location

Vicente Bolea Sanchez and Beomseok Nam
School of Electrical andComputer Engineering
Ulsan National Institute of Science and Technology
Ulsan, Korea, 689-798
Email: vicente.bolea@gmail.com, bsnam@unist.ac.kr

***Abstract*—In this work, we present a novel distributed file processing framework called *UniDQP* that takes into account the dynamic nature of file I/O request distribution and remote contents of distributed caching infrastructure. The job scheduler of the framework makes scheduling decisions based on the spacial location of the queries. In addiction, we propose an innovate scheme enhancing the cohesion of system.**

I. Introduction

Distributed and parallel query processing middleware systems have been used for decades to solve large and complicated scientific problems as substantial performance gains can be achieved by exploiting data and computation parallelism. MapReduce framework has recently gained tremendous popularity as a key distributed and parallel query processing framework due to its scalability and easy programming model []. However, MapReduce framework is not designed to leverage the large amount of memory space available in the back-end distributed servers since its target applications are mostly performing one time data analysis. Unlike the web data processing applications scientific datasets are often reused by multiple jobs and semantic caching plays an important role in improving the system throughput and job response time. Another design decision that we have to make concerens is the data skew challenge in MapReduce framework. In [**?**], [**?**], [**?**] it has been reported that some map tasks (stragglers) take significantly longer than the average execution time of other map tasks. In order to mitigate the stragglers problem, MapReduce framework runs some backup tasks to alleviate the problem, however the back-up tasks do not help when the input datasets are not well balanced.

In this work, we present a distributed and parallel query processing middleware framework - *Orthrus* which takes into account the large memory space in back-end servers and makes scheduling decisions based on the cached data objects in the memory instead of data file location. Orthrus is a two layer architecture as it deploys a layer of distributed semantic caches on top of a distributed file system layer. In order to balance the system load and to improve data reuse rate, the front-end scheduler of Orthrus needs to estimate the cached contents in distributed semantic cache layer for data reuse, and assign equal amout of jobs to each server for load balancing. A challenge in estimating the remote cache contents in distributed query processing framework is that the front-end scheduler should exchange large amount of information about remote caches as the cached objects in each server change very fast as they process jobs. Capturing a global snapshot of remote servers' cache contents is certainly a very expensive operation especially in a large scale system. Another challenge in designing cache-aware query scheduling policies is that the scheduling algorithm must be light-weight so that it doesn't bottleneck the front-end scheduler.

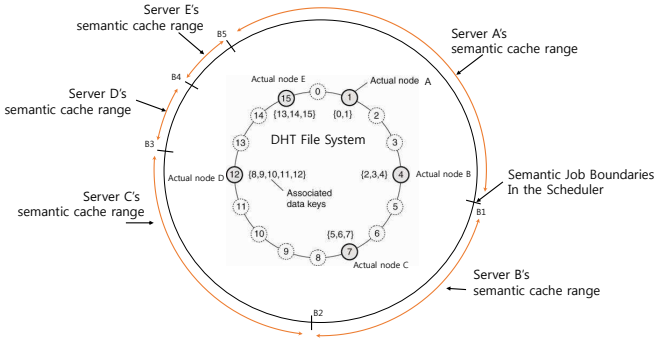

Fig. 1. *Two layers structure of back-end servers.*

In this poster presentation, we present the two layer architecture of our distributed and parallel query processing framework, its scheduling algorithms, data migration policies, and preliminary results of performance evaluation.

II. Orthrus: Distributed Query Processing Framework

We propose a framework composed of a set of accesible front-end servers and a collection of back-end caching servers organized in a two layers peer-to-peer structure.

1) Inner layer: It consists in a set of not mutable static segments which represents the range of the dataset corresponding to each back-end server. This layer is visible from every back-end servers throught its hashing lookup table.

2) Outer layer: Analogously, we propose a non-static layer divided in mutables boundaries which represents the current

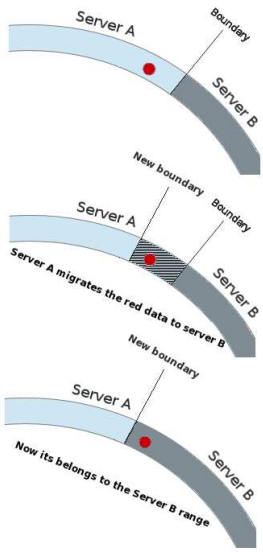

Fig. 2. *Two layers structure of back-end servers.*

status of the cached data. This layer is not completely visible, where in contrast to the inner layer, each back-end node only knows its boundary and neighbors. Those boundaries are in a continous movement due to our spacial algorithm.

A. Data migration

An intensive use of data migration is necessary to guarantee a high cohesion between the front-end and back-end servers. Our framework accomplish two main problems making use of this policy:

1) Overlap of the boundaries: Due to a continous movement of the boundaries of the outer layer, at some point the data located near by its boundary could be out of its original range, for this reason our framework corrects the overlap migrating overlapped data to the range where it belongs.

2) Requesting data: As the ranges are continously moving, the scheduler can not know accurately which back-end node contains the requested data. In order to overcome this problems our framework provide a hashing lookup table to each back-end server indicating which range in the inner layer contains the requested data. In case of failure of the scheduler, the requested data will be readed from the corresponing back-end server and cached in the back-end server selected by the scheduler.

III. Experiments

IV. Conclusion

References