A Recommender for Choosing Data Systems based on Application Profiling and Benchmarking

Elton Figueiredo de Souza Soares, Renan Souza, Raphael Melo Thiago, Marcelo de Oliveira Costa Machado, Leonardo Guerreiro Azevedo

[1]IBM Research
Rio de Janeiro – RJ – Brazil

{eltons,marcelo.machado}@ibm.com, {rfsouza, raphaelt, lga}@br.ibm.com

Abstract. *In our data-driven society, there are hundreds of possible data systems in the market with a wide range of configuration parameters, making it very hard for enterprises and users to choose the most suitable data systems. There is a lack of representative empirical evidence to help users make an informed decision. Using benchmark results is a widely adopted practice, but like there are several data systems, there are various benchmarks. This ongoing work presents an architecture and methods of a system that supports the recommendation of the most suitable data system for an application. We also illustrates how the recommendation would work in a fictitious scenario.*

1. Introduction

In our data-driven society, there are hundreds of possible data systems in the market, each with a large variety of characteristics, *i.e.*, a large variety of configuration parameters to be adjusted to fit applications requirements. Thus, it is challenging for enterprises and users to find the most suitable data systems to use [Brahimi et al. 2016]. This problem is even worse due to the lack of representative empirical evidence to help users to make an informed decision in their choice of data systems based on their specific requirements.

The use of benchmark results is a widely adopted practice to base a decision [Gray 1992], but there are several data systems and many benchmarks [Huppler 2009]. Usually, enterprises and users spend days performing the data system search, *e.g.*, performing the following steps: (i) Choose a set of data systems; (ii) Search for a suitable benchmark for evaluation; (iii) For each data system, define the most suitable configuration; (iv) Run the benchmark to find the most appropriate data system; (v) Analyze the results. These steps are very time-consuming and error-prone if not executed in a very systematic way.

In this ongoing work, we propose an architecture and methods of a system that recommends the most suitable data system and its configuration parameters by means of benchmark results and application profiling. Our approach: (i) Automatically identifies and characterizes data systems and benchmarks; (ii) Generates users' profiles by analyzing how their applications access the current used data system; (iii) Recommends the most suitable set of benchmarks' and candidate data systems' configurations based on the application profile; (iv) Iteratively evaluates the minimum set of candidate data systems' and benchmarks' configurations required to identify the top-k data systems; (v) Presents an ordered list of the top-k candidate data systems for the user. The proposal's primary goal is to support enterprises and users, saving significant amounts of time, resources, and effort in finding benchmarks and data systems for their application needs.

In an extensive literature review, we did not find any work that accomplishes this goal. Existing approaches do not automatically characterize application access patterns and use data systems and benchmarks configurations. Furthermore, they do not recommend benchmark configurations that best represent a specific user application and tuning of a particular data system, regardless of the user application.

The remainder of this work is organized as follows. Section 2 presents our proposal architecture and methods. Section 3 illustrates its use in a fictitious application. Finally, Section 4 concludes this paper. Due to the lack of space, details about the related work evaluation, background and sub-process are presented in the appendix[1].

2. AI-Enhanced Advisor

The system architecture is presented in Figure 1. On the left, it is depicted the application and the data system accessed by it. The proposed system captures the data access patterns used by the `User Applications`. Data access patterns correspond to the percentage of data accesses classified as belonging to a data access pattern class, *e.g.*, `[ ("Deep Traversal Query", 30%), ("Large Bulk Insertion", 50%), ("Short Analytical Query", 20%)]`. These data access patterns are used to characterize the user application profile.

The system architecture is composed by the following components:

- `User Interface (UI)`: Supports user interaction features, *e.g.*, create recommendation request, visualize top-k candidate systems and evidence.
- `Application profiler`: Generates specific application profiles by analyzing how applications access their data systems.
- `Benchmark Advisor`: Periodically, characterize benchmarks through crawling several data sources, *e.g.*, digital libraries of scientific papers, remote git repositories, and websites in general.
- `Data System Advisor`: Recommends benchmarks' configurations and candidate data systems' configurations based on the application profile.

The data used by the system is stored in the following databases:

- `Application profiles`: Stores data access patterns and applications deployment configurations.
- `Benchmarks' characterizations`: Stores a set of queries, inserts, updates, and deletes statements used in benchmarks along with their parametrizations, metadata of benchmark metrics, formulas, and any other additional information required for running and parameterizing the benchmarks.
- `Data Systems' characterizations`: Stores data systems, their configurations, and each configuration's fitness to each data access pattern class.
- `Policies`: Stores a list of rules and policies that the company uses regarding the selection of any data system, *e.g.*, "All third-party systems must be open source and have online support".

[1] `https://ibm.box.com/v/sbbd-2021-data-system-advisor`

- `Advisor results`: Stores the results of benchmark executions and final scores generated by the advisor for ranking the top-k data systems recommended for the given application profile.

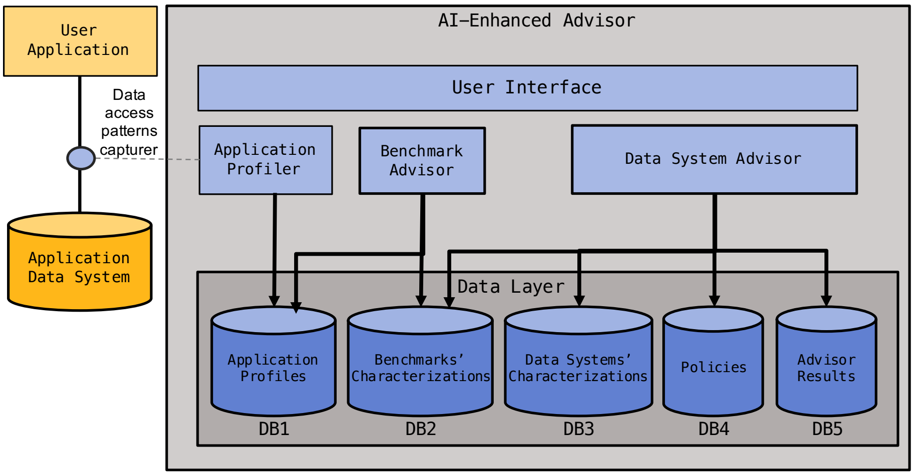

Figure 1. Architecture Overview

The method supported by the system is presented in Figure 2. It starts when a *Data system recommendation request* is received. Then, the `Application Profiler` generates the profile of the application to characterize its data access patterns.

After receiving the application profile from `Application Profiler`, the `Data System Advisor` queries the `Advisor Results` database using the data access patterns from the application profile as input. Based on executions of benchmark configurations, if it finds results generated that simulate similar data access patterns, then it returns the top-k data system configurations, *i.e.*, instead of going through all intermediate steps, it presents the results to the user, and the process ends.

Otherwise, the `Benchmark Advisor` queries the `Benchmarks' Characterizations` database looking for parts of benchmarks' application profiles that contain some data access patterns of the target application profile. Both kinds of profiles have the same structure, *i.e.*, they are composed of a list of data access pattern percentages. The result is a set of benchmarks that support the evaluation of data systems considering the application needs. If no match is found, the method ends.

Otherwise, the `Data System Advisor` gets the data system configurations that match the application profile from the `Data Systems' characterization` database sorted by fitness values. Then, the `Data System Advisor` queries the `Policies` database to filter data systems that are not under the company policies. Besides, specific policies may also be explicitly defined for a given application, which should be registered in the `Policies` database. Examples of policies are: all applications must be developed using open source technologies; all application licenses must allow commercial use.

Afterwards, the `Data System Advisor` searches for benchmark configurations' results that can be applied to get the top-k data systems configurations for the recommendation. If the resulting information is enough, the `Data System Advisor` generates the top-k data systems configurations and presents to the

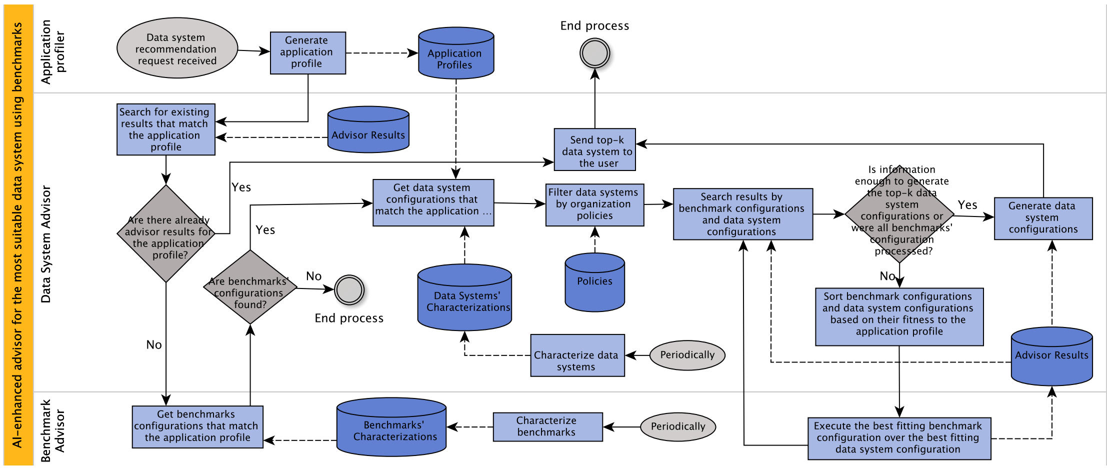

Figure 2. Method supported by the system

user. The top-k data system configurations are generated based on the benchmark results. Each benchmark configuration execution with each data system configuration generates one `Score`. The fitness of this benchmark configuration weights the score of each benchmark configuration to a data access pattern as `WeightedScore = Score x BenchmarkConfigurationFitness`. This weighted score is then multiplied by the proportion of data accesses that present a specific data access pattern class weighted by the expected cost of each access of this class, which is a function of the hardware consumption and average time of each access, which is `DataAccessPatternScore = WeightedScore * (DataAccessPatternProportion * f(ResourceConsumption, AverageTime))`. Therefore, the final score obtained by a data system configuration for a given application profile (*i.e.*, DataSystemConfigurationScore, is the sum of the `DataAccessPatternScores` of each data access pattern class contained in the application profile. The top-k data system configurations are identified by sorting the configurations by `DataSystemConfigurationScore` in descending order and filtering the first k configurations retrieved.

If there is not enough information to generate the top-k data systems, the `Data System Advisor` sorts the benchmark configurations and data systems configurations based on their fitness to the application profile. It executes the best appropriate benchmark configurations over the best fitting data system configurations until it gets the information needed to generate the top-k data systems configurations or has processed all the benchmarks. In the last case, less than top-k data system configurations are returned, or no data system configuration is found. The `Data System Advisor` generates a set of data systems that meet the application profile using the benchmark results and the filtered data system configurations,

The method has sub-processes to collect the required data periodically, giving support so that the system can work on up-to-date data (`Characterize data systems` and `Characterize benchmarks`), and sub-processes to `Generate application profile` and to `Get data system configurations that match the application profile` which are not here due to lack of space.

3. Exemplary Scenario

This section illustrates the use of the proposal in a fictitious scenario. Consider a *Law Interpretation Advisor* application implemented initially using *Oracle 11g*. It uses Natural Language Processing techniques to extract knowledge from legal documents and advises users to interpret specific legal topics. It supports the update of the knowledge extracted via advice feedback. The application administrator has to decide if the current data system is the best suitable option for this application, and uses our proposal for this assessment.

1. The administrator requests a data system recommendation through the `User Interface` for the *Law Interpretation Advisor* application.
2. The `Application Profiler` starts the monitoring of the data accesses of the *Law Interpretation Advisor* to the *Oracle* database, by intercepting the API calls through the network or capturing the database logs from disk, and uses AI techniques to extract patterns from the data accesses (*e.g.*, query clustering [Morsey et al. 2011], decision trees [Elnaffar et al. 2009], etc.).
3. Generated *application profile* is stored in `Application Profiles` database.
4. The `Benchmark Advisor` gets the *application profile* and queries the `Advisor Results` database looking for results that assessed systems with the same *application profile* as the current profile of *Law Interpretation Advisor*.
5. As no result is found, the `Benchmark Advisor` queries the `Benchmark Configurations`, and it finds benchmarks to be used in the analysis, *e.g.*, LUBM100 [Guo et al. 2005] and DBPSB [Morsey et al. 2011]).
6. The `Data System Advisor` gets the *data system configurations* that are expected to provide the best performance of each data system for the given application profile from the `Data Systems' characterization` database, *e.g.*, *DB2 version 11 with Partitioned Indexes* [Zilio et al. 2004], *DB2 version 11 with Clustered Indexes* [Zilio et al. 2004], *Oracle 11g with b-tree indexes* [Kuhn et al. 2012], and *Oracle 11g with bitmap indexes* [Kuhn et al. 2012]).
7. Then, the `Data System Advisor` gets the policies from `Policies` database, *e.g.*, the policy *'Only data systems with 24h support may be used in the company'*, and filters data systems. In that case, no data system is filtered.
8. The `Data System Advisor` identifies that there is not enough information to get the top-k data system configurations. It sorts the data systems and benchmark configurations and sends them to the `Benchmark Advisor`.
9. The `Benchmark Advisor` receives the configurations and performs all steps required for the execution of each benchmark in each Data System configuration, such as data system deployment (*e.g.*, spawning containers and automated jobs in a Kubernetes cluster), data set loading, query execution, and performance metric collection. It stores the generated results in `Benchmark Results`.
10. The `Data System Advisor` checks that it has enough information and gets the top-k most suitable data system configurations, *e.g.*, *DB2 version 11 with Partitioned Indexes* [Zilio et al. 2004] and *Oracle 11g with b-tree* indexes [Kuhn et al. 2012]. This final ranking is performed using techniques like a weighted average of the scores obtained on each of the benchmarks.
11. The `Data System Advisor` stores the top-k data system configurations in the `Advisor Results` database, and it sends them to the user.
12. The user access the `User Interface` to see the report of the final recommendation and detailed explanation of the whole process executed by the advisors.

4. Conclusion

The deployment of high-performance systems is a requirement in our society. However, technology evolves, systems are maintained according to new requirements, and the performance usually degrades. A critical aspect of improving performance in modern software systems is to use an efficient data system for the application, but choosing one among several options is not trivial. Often users rely their choices based on benchmark results. Nevertheless, just like there are many data systems, there are several benchmarks. Therefore, choosing data systems (with their several configurations) and the proper benchmarks to evaluate them is a challenge which we addressed in this ongoing work.

Existing works do not automatically characterize application access patterns and data systems' and benchmarks' configurations. They cannot recommend benchmarks' configurations that best represent a specific user application and focus on automated tuning of a particular data system, regardless of the user application. In contrast, we focus on optimizing the comparison of multiple data systems. We proposed an architecture and methods that address these issues and exemplified its use in a representative scenario. With our proposal, enterprises and users would save effort to find the most suitable benchmark and the best data system – consequently, saving time and resource. As future work, we are working on the implementation towards its evaluation in a real application. In an initial implementation, we developed a benchmark executor based on a configuration file. It enables customization for a given application and automatically executes the benchmark on the available hardware and summarizes the results through charts.

References

Brahimi, L., Bellatreche, L., and Ouhammou, Y. (2016). A recommender system for dbms selection based on a test data repository. In *East European Conference on Advances in Databases and Information Systems*, pages 166–180. Springer.

Elnaffar, S., Horman, R. W., Lightstone, S. S., Martin, P., Schiefer, B. K., and Van Boeschoten, R. D. (2009). Method for identifying a workload type for a given workload of database requests. US Patent 7,499,908.

Gray, J. (1992). *Benchmark handbook: for database and transaction processing systems*. Morgan Kaufmann Publishers Inc.

Guo, Y., Pan, Z., and Heflin, J. (2005). Lubm: A benchmark for owl knowledge base systems. *Journal of Web Semantics*, 3(2-3):158–182.

Huppler, K. (2009). The art of building a good benchmark. In *Technology Conference on Performance Evaluation and Benchmarking*, pages 18–30. Springer.

Kuhn, D., Alapati, S., and Padfield, B. (2012). *Expert Indexing in Oracle Database 11g: Maximum Performance for your Database*. Springer.

Morsey, M., Lehmann, J., Auer, S., and Ngomo, A.-C. N. (2011). DBpedia SPARQL Benchmark–Performance Assessment with Real Queries on Real Data. In *International semantic web conference*, pages 454–469. Springer.

Zilio, D. C., Rao, J., Lightstone, S., Lohman, G., Storm, A., Garcia-Arellano, C., and Fadden, S. (2004). DB2 Design Advisor: Integrated Automatic Physical Database Design. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 1087–1097.