E2C-Sim-DB: Database Management for Scheduling Policy Simulator

CMPS 460- Database Management Systems

Drake Rawls[1], Connor Rawls[2]
C00211180[2]
12/2/22

Abstract

Databases and database management can often be considered the entry point to a new project. A database's ability to stand as the foundation for a project can be recognized as its intelligent design and management, or conversely, lack therof, radiates throughout the project entirety. Due to the imperative nature of such a component in a project's architecture, it is important to fully understand how to construct a database properly and how to wholly manage its internal machinations. In this paper, the authors present a database for a currently-implemented scheduling algorithm simulator built for illustrating the behavior of incoming tasks to a system. As such, this paper illustrates the design process and implementation for the final product.

I. Project Requirements

Background

Oftentimes, during the development cycle of a new architecture in the realm of cloud, the developers would like to prototype how their system handles incoming tasks. For scalability purposes, a scheduling or load balancing policy is emplaced to assure that system resources are utilized efficiently. Under this scenario, E2C-Sim is a scheduling policy simulator designed for load-testing edge-to-cloud environments with differing scheduling algorithms, machine heterogeneity properties, and task characteristics [1].

E2C-Sim implements a large set of various data to highlight the interactions between system configurations and the workload provided. For the rest of this paper, we will simply refer to the theoretical system environment that E2C-Sim portrays as the system environment. The state of a given system environment is composed of the available *task types* that can be provided to the application machines, *machine types*. Each task type possesses a set of characteristics such as *detail*, a specific task's deadline, and *urgency*. We will use the context of web requests to explain the nature of a task's deadline and urgency. Under the scenario in which a user might send a request to a website, the user can expect to receive a response within an ample amount of time. Otherwise, the user might grow unsatisfied with the web service or, in the worst-case scenario, leave the website entirely. The deadline can be defined as the hard limit in which a task (web request) is expected to complete. The urgency of a task characterizes the importance of such a task. In a website, some pages or links might be recognized as more urgent than others. Therefore, a task's priority can be utilized by a scheduling algorithm to properly decide how it should be handled. One can observe how a given workload might be composed of many different

task types. Likewise, the given application machines that answer such tasks can also be heterogeneous in nature. Each machine type can be categorized by various features such as idle power consumption, number of processing cores, and available memory. Given such a mixture of task types and machine types, it can be expected that a given task might behave differently if executed on one machine as compared to if that same task were to be executed on a different machine of a different machine type.

From this assumption, E2C-Sim derives the *EET*, Expected Execution Time, matrix. This is a collective of preprofiled information containing each task type's expected execution time for each machine type. In the realm of databases, one might think of such a table as a cartesian product between a table containing task type and machine type information, with the entry values for each index in the EET being the expected execution time. Further leading into E2C-Sim's functionality, each of these tables are utilized to generate varying loads to test a given system environment that is composed of given task types and given machine types.

Maintaining the analogy of web applications, the behavior of incoming user requests can prove quite volatile. During times in which most users would not be using a website, an incoming workload may possess sparse requests. Comparatively, during exemplary periods such as Black Friday, a website might see large bursts of incoming user requests. To imitate the nature of varying workloads, E2C-Sim maintains *scenario* data. Under a single scenario, the system might see tasks of different types, start times, and distributions. In general, a scenario can be thought of as the overencompassing behavior of a given set of tasks.

From a given scenario, a finalized *workload* can be generated. The workload examines each entry in a scenario and calculates the task instance arrival times. For example, if a given scenario is composed of task types T1 and T2, each with a total number of 50 instances and exponential distributions, respectively, the workload will be propagated with 100 total task instances ordered by arrival time. The arrival time for each instance is formulated from each scenario entry's start time, end time, and distribution (exponential, logarithmic, etc.).

User Case

Being that E2C-Sim possesses such a large set of complicated data, a database alongside an accompanying database manager is to be implemented to facilitate the interaction of its data. Each aforementioned set of data, task type, machine type, and eet, is to be stored as a base table to be referenced further. The scenario table is something to be generated from a given file

provided by the user. Once the scenario file is loaded into the manager, the user will have the ability to generate a specified number of workload tables from the single scenario table. The purpose of generating multiple workload tables is up to the prerogative of the user. For the sake of validity, users might wish to perform such an experiment multiple times and examine results, such as deviation, to determine if their results are conclusive or not.

Specifically, the user will interact with the manager of this project as follows:

1. Launch the database manager.
2. Load in a scenario file.
3. Input the number of workload tables to be produced.
4. Generate the workload tables and view their contents.
5. Interact with the data through various utilities.

II. Data Modelling

It can be considered that the use of data by E2C-Sim starts with the task type and machine type datasets. This information is something that remains static between experiments and only changes once the user's test environment changes (adding a new machine, removing a web page, etc.). These two datasets are stored in the aptly named *task_types(task_id[PK], name, detail, urgency)* and *machine_types(machine_id[PK], no_of_replicas, idle_power, max_power, num_of_cores, cpu_clock, memory)* tables.

Based upon the environment's task types and machine types, the derived EET table is developed. This table is dependent on the available tasks and machines, changing with the insertion/removal of entries from the previously described tables. The EET table can be defined as *eet(task_id[FK], machine_id[FK], eet)*.

The scenario table is propagated once the user loads a given scenario file. The reasoning for this interaction is due to the will of E2C-Sim's creator. It is preferred that each scenario is stored as a separate file, rather than existing as a single entry in a singular table. The scenario table possesses the attributes *scenario(task_id[FK], dist_id[FK[, start_time, end_time, num_of_tasks)*.

The *dist_id* attribute in the scenario table is a foreign key from another descriptive table, *distribution(dist_id[PK], name)*. Simply, this table is implemented for security reasons. In past cases, it was found that users would claim different names for the same distribution trend. As such, it was decided that a preventative measure be emplaced to combat potential future errors.

Users have the ability to observe the available distributions supported by E2C-Sim and their corresponding *dist_id*.

Lastly, the table *workload(instance_id[PK], task_id[FK], name, arrival_time)* is generated from the scenario table. The purpose of this table is to provide the simulator with a workload of incoming tasks, characterized by each task instance's arrival time. It should be noted that a task instance is a singular occurrence of a task of a given task type.

The overlying ER diagram for this project's database can be found in Figure 1.

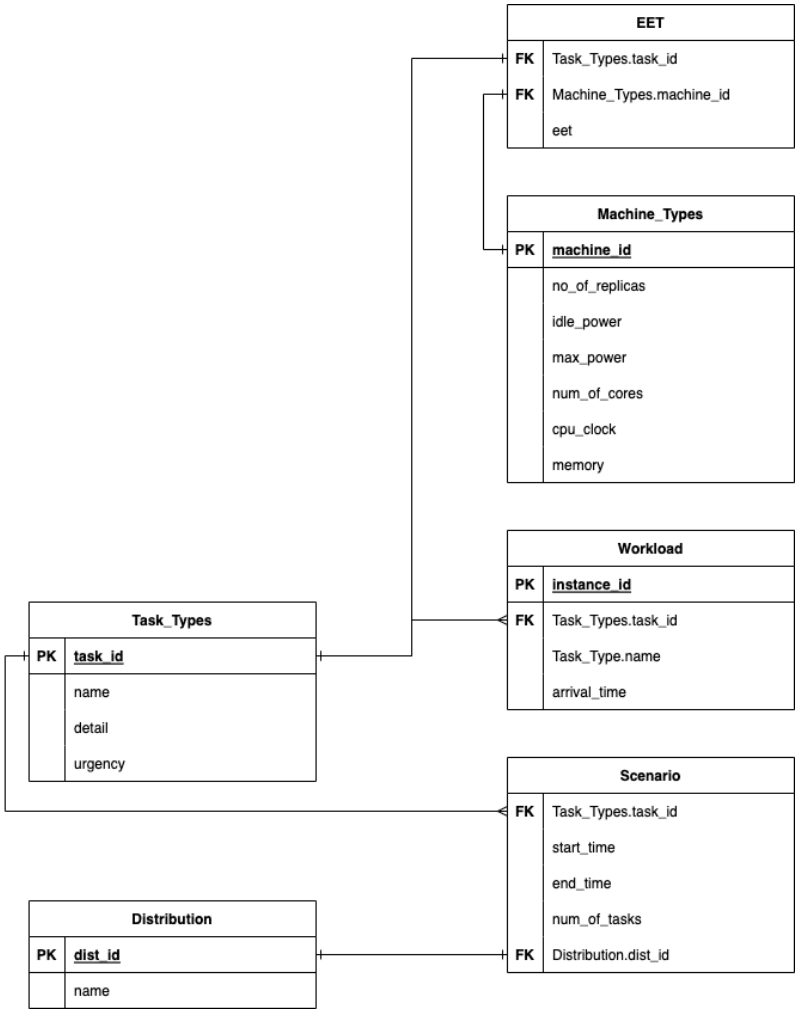

Figure 1: ER Diagram for E2C-Sim-DB

Normalization

Before continuing with the implementation, the normalization of the database should be discussed. The authors of this paper recognize that the database is not in Third Normal Form (3NF). In fact, the database is currently in 1NF. This is due to the inclusion of *task_types.name* in the workload table. If this occurrence were not prevalent, then the database would be recognized as satisfying the condition for 2NF: “Non-key attributes should not have functional dependency to any subset of the primary key.” Furthermore, the 3NF conditions of 1. “The table must be in 2NF” and 2. “There is no functional dependency between non-key attributes” are also satisfied.

With these understandings being said, it should also be mentioned that the current scheme for the database is implemented this way to better facilitate E2C-Sim’s current functionality. The refactorization of E2C-Sim and the database could be considered as a future work.

III. Implementation

Due to time constraints, E2C-Sim-DB is currently implemented as a standalone application. While our manager is not embedded into the parent application, it is a working prototype that models E2C-Sim’s actual structure and utilizes data from the real simulator.

E2C-Sim-DB utilizes the SQLite language through the Python library *sqlite3*. Additionally, all management modules were developed in Python. A list of all essential Python modules used in the application can be observed Table 1.

Library	Purpose
sqlite3	SQLite language facilitator
pandas	Array manipulation
numpy	Array manipulation
PyQT5	Graphical User Interaction

Table 1: Python Libraries Used for E2C-Sim-DB

The main service is started by calling the *main.py* program. This file launches the graphical user interface for the user to interact with. After the GUI has loaded, an initialization function is called to check the current database file that is referenced by the service. If the database does not possess the recognized base tables/schemas (task types, machine types, and EET), it will generate new tables for each and fill them with their respective default values.

From here, the user can select which scenario file to be loaded into the manager. In the source code, once the user prompts which file to be used, the current scenario table that resides in the database file is compared with the new scenario data. If the data is not concurrent, then the current scenario table is truncated and the new scenario data is imported into the scenario table. This feature prevents unnecessary writing to the database while allowing the user to change scenarios on the fly. The last action required by the user to present anything meaningful is for them to enter in the desired number of workload tables to be generated. Once the user clicks the generate button, an in-house program is triggered to observe the scenario entry characteristics. Each entry possesses a task type, start time, end time, number of tasks, and distribution scheme. The number of tasks is sampled from the given distribution utilizing the start and end times as the possible range, with the output being the task type and the arrival time of the instance. Once all of the task instances have been generated, the workload is ordered in accordance to the arrival time of each task. This ordering allows for E2C-Sim to simply pull from the workload table and treat it as an incoming queue of tasks. It will repeat this process the desired number of times as the user prompted the manager for. A random number generator is utilized as a seed in the distribution sampling to ensure that each table has a low probability of being duplicated.

IV. Security Features

To ensure the security of this database, a number of measures are emplaced. Firstly, the user is prompted for a username and password upon entry. This determines if the user is recognized by the application to possess access to the data and continue operation. Secondly, users only have the ability to alter certain tables in the database. For example, the EET table is derived from the task types and machines types tables. The user is not allowed to alter the EET table directly, as to prevent any loss of necessary information for the simulator. However, if the user does include a new entry in the task types or machine types table, the EET will be updated through the manager. Additional to these security measures, users are prompted before any deletion of a tuple from a table. A simple “Are you sure?” box can prove decisive for if a month’s work of experimentation is gone with the wind.

V. Other Features

The implementation of E2C-Sim-DB was done with ease-of-use, isolation, and modularity in mind. Rather than provide access to the database through means of sockets and web applications, it was decided that using SQLite provided ample availability. Furthermore, the

manager launches through its own GUI instead of requiring a web browser. What these features entail is that one can simply port the program from a distributor such as GitHub and launch it immediately without any previous knowledge of how database servers operate. All of the required modules and files are encapsulated as one package. The only work prior to operation that a user needs to put in is the different scenarios that they wish to examine the behavior of. This accessibility allows users to abstract the underlying complications needed to run the program, while providing the freedom to customize their experiment.

I. References

[1] E2C-Sim, Edge-to-Cloud Simulator. Online: https://github.com/hpcclab/E2C-Sim/tree/main. [Online; 2022-12-1].