2014 14th International Conference on Computational Science and Its Applications

Distributed Enviroment Framework for Optimization Experiments

Pedro Abreu
INESC TEC
Faculdade Engenharia
Universidade Porto

Carlos Soares
INESC TEC
Faculdade Engenharia
Universidade Porto

Rui Camacho
LIAAD
INESC TEC
Faculdade Engenharia
Universidade Porto

***Abstract*—Optimization studies often require very large computational resources to execute experiments. Furthermore, most of the time, the experiments are repetitions (same problem instances and same algorithm with the same parameters) that were carried out in past studies. In this work, we propose a framework for the execution of optimization experiments in a distributed environment and for the storage of the results as well as of the experimental conditions. The framework can support not only the organized execution of experiments but it also enables the reuse of the results in future studies.**

I. Introduction

Optimization studies, such as the ones carried out in fields like Algorithm Selection [1], Fitness Landscape [4] and Algorithm Benchmark [7], require data about the optimization instances, the optimization algorithms and the results of the application of the algorithm to the instance. To gather the data needed for those studies large computational resources are required, due to the nature of the optimization problems and the existing algorithms. In addition to the computational power, it is necessary to store all data to be analyzed in the study. Depending on the type of study it might be also necessary to store information about the problem instances, performance results, solutions, algorithm parameters, execution logs, etc.

There are well developed systems that store, organize and share that data in other scientific areas like Machine Learning, Bioinformatics, Astronomy and Physics. For instance, openML (http://expdb.cs.kuleuven.be) is a collaboration framework designed to easily share machine learning experiments with the community and automatically organize them in public databases [5].

We propose a new framework to support optimization studies. This framework enables the execution of optimization algorithms in a distributed computational setting supported by a database to store and distribute the data. The schema of the database is designed taking into account that its purpose is analytical, not transactional [2]. Therefore, there is a particular concern for enabling fast access to the data.

The current state of the framework is as follows: There is a transactional database populate with Job-Shop Scheduling (JSS) instances and runs with heuristic algorithms for JSS and Genetic Algorithms. The current system allows to schedule and run the experiments in a distributed enviroment such as the called IBERGRID. There is a database for analyse purpose with performance data about the algorithms and features concerning the JSS for Algorithm Selection studies [1].

The rest of the paper is structure as follow. In Section II, we describe the JSS problem for a more detail understanding. The framework is describe in Section III

II. The Job-Shop Scheduling Problem

The deterministic job-shop scheduling problem can be seen as the most general of the classical scheduling problems. Formally, this problem can be described as follows. A finite set J of n jobs $\{J_1, J_2, \ldots, J_n\}$ has to be processed on a finite set M of m machines $\{M_1, M_2, \ldots, M_m\}$. Each job J_i must be processed once on every machine M_j, so each job consists of a chain of m operations. Let O_{ij} represent the operation of job J_i on machine M_j, and let p_{ij} be the processing time required by operation O_{ij}.

The operations of each job J_i have to be scheduled in a predetermined given order, i.e. there are precedence constraints between the operations of each job J_i. Let $\prec$ be used to denote a precedence constraint, so that $O_{ik} \prec O_{il}$ means that job J_i has to be completely processed on machine M_k before being processed on machine M_l. Each job has its own flow pattern through the machines, so the precedence constraints between operations can be different for each job. Other additional constraints also have to be satisfied. Each machine can only process one job at a time (capacity constraints). Also, preemption is not allowed, so operations cannot be interrupted and must be fully processed once started. Let t_{ij} denote the starting time of operation O_{ij}. The objective is to determine starting times t_{ij} for all operations, in order to optimize some objective function, while satisfying the precedence, capacity and no-preemption constraints. The time when all operations of all jobs are complete is denoted as the *makespan* $C_{\max}$. In this paper, we consider as objective function the minimization of the

978-1-4799-4264-0/14 $31.00 © 2014 IEEE
DOI 10.1109/ICCSA.2014.59

makespan:

$$\begin{aligned} C^*_{\max} &= \min\left(C_{\max}\right) \\ &= \min_{\text{feasibleschedules}}\left(\max\left(t_{ij}+p_{ij}\right)\right), \\ &\quad \forall J_i \in J, M_j \in M. \end{aligned}$$

A comprehensive survey of job shop scheduling techniques can be found in [3].

III. Framework Architecture

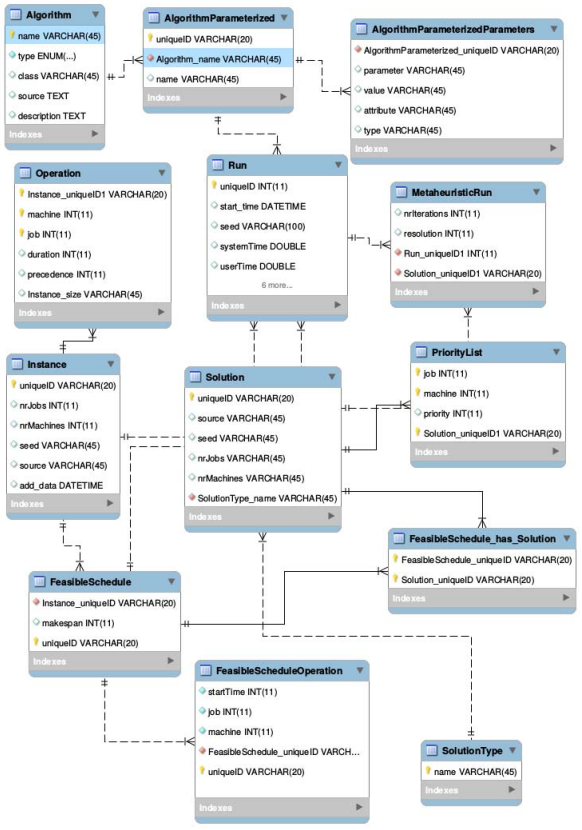
In this section, we describe the databases used to store data and the general architecture of the framework.

A. Database schema

This system has two databases with different schema and different objectives. One database is an operational database, optimized for algorithms execution: get and store the data. This database is called Optimization DB. The second database is used to obtain data for analysis purposes. This is called Analysis DB. The data in this database is obtained from the Optimization DB.

1) Optimization Database: The main purpose of the Optimization DB is to collect all data necessary the execution of the experiments. So, type of schema of this database is normalized relational database. This type of database for insert, update and delete operations maintaining the consistence (Figure 1).

Figure 1. Schema of the Optimization Database



Algorithm: The algorithms applied are a parametrized instance of an general algorithm like Genetic Algorithm, Giffler And Thompson, Simulating Annealing, etc. Each parametrized algorithm is store in the AlgorithmParameterized table that is related to the Algorithm table that is the general algorithm information is stored. The parameters values are store in the AlgorithmParameterizedParameters table.

Instance: Each instance identification is store in the Instance table with the size (number of jobs and machines), the seed (if random generated) and the source (since can be an well know instance like Taillard [6]). All information concerning the operations of the instance are stored in the Operation table. The operation is identified with the job and machine index, duration and precedence.

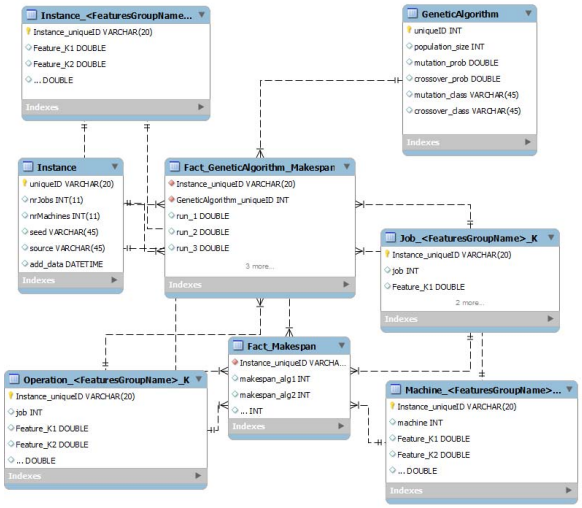
Schedule: Each schedule identification is stored in FeasibleSchedule table with the instance identification that belongs to. The start times of each operation are stored in FeasibleScheduleOperation table. This table has the identification of the schedule that the operation belongs, the machine and job index and the start time.

Run: Every parametrized algorithm get, as input, an instance and returns a schedule. The id of the instance, parametrized algorithm and the schedule are stored in the Run table. The status field gives information about the current state of the execution. The possible values are: "None" if the experiment is schedule, a string value that is a unique id of a execution been taken in some machine and "Finish" means that the experiment was executed with success.

Metaheuristic Run: The Metaheuristics are algorithms that have several iteration, in each, one or more solutions are generated. In each iteration, the algorithm evaluates one or more solutions until a stop criteria is satisfy. These solutions can be store in the database. Each type of solution (depends of the problem and the algorithm) has a specific table. In the current database, exists one type of solution, called the Priority List solution, that have a specific table. In this table, they are stored with an identification of solution abstract identification associated to the Solution table. The table Metaheuristic has the identification of the solution evaluated and the iteration number in which the solution has been generated.

2) Analysis Database: The Analysis DB is designed to feed the optimization studies with the necessary data. The Optimization DB does not have the information ready to be analysed such as the algorithm performance measures and instance features. All this information must be calculated. Since it is a calculation that is done often and never changes so it is more efficient to store. The Optimization DB isn't design to a fast and consistence analyse. Since problems have little in

Figure 2. Schema of the Analysis Database



common, each optimization problem has its own analysis database, called <optimization problem>_Analysis. Figure 2, shows the JSS_Analysis schema.

The schema of JSS_Analysis is a Fact Constellation Schema. The Fact Constellation has several Fact tables and each one has one or more Dimension tables. Each Fact table can share the same Dimension table with other Fact table. A Fact table have the measures that are the target of the study connect by a Foreign Key to the Dimension tables that have attributes about the object of the study [2].

Dimension Tables: The dimension tables of type Instance/Machine/Job/Operation_GroupFeatures have features that describe the JSS instances.

The Instance table allow a better control of the instances used in a study, with the data inserted in the analyse database (allow to the analyst always use the same set of instance even if there are new instances), the size (number of jobs and machine) and the source of the instance (random or a well know instances for the community).

The dimension table *GeneticAlgorithm* stores the parameters values for each genetic algorithm executed.

Fact Tables: The fact table *Fact_Makespan* stores data about the makespan obtained by the algorithms. It supports the comparison between their performance under the different features values represented in the dimensions tables.

The fact table *Fact_GeneticAlgorithm_Makespan* stores the makespan of each execution when applying the genetic algorithm, with the same parameters, to an instance. In addition to the comparation between the makespan under different features values, also can be compare to the different parameter values stored in dimension table *GeneticAlgorithm*. If there are other algorithms for variation parameter studies like *Simulated Annealing* algorithm then a new fact table should be created and other dimension table with the algorithm specific parameters.

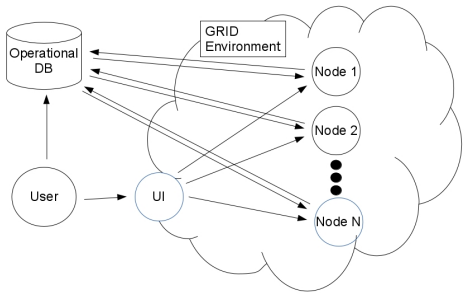
B. Distributed Environment

The computational problem of the execution of many algorithms for many instances can be solved with the use of distribute computation. They are executed in several machines in a GRID environment, in this case the IBERGRID.

The GRID computation paradigm has the purpose to integrate several computation resources that can belong to independent organizations. The user should does not need to be aware about the different systems of the resources and the task of distributing the jobs into the GRID structure. This way a high performance infrastructure should emerge in dispersed resources with a unique interface to the user.

Figure 3, shows the communication schema between all elements of the framework.

Figure 3. Communication schema of the framework



The elements of this framework are:

Database Server: The database server stores data concerning the experiments to be execute, the optimization problem, the algorithm parameters and the results about execution.

GRID: The communication of the GRID with the user is using the User Interface (UI). Accessing the UI using a ssh connection, the user can launch several jobs into the GRID enviroment that execute the scripts. This scripts do not have information about the experiment to execute and do not store the results in User Interface. All input and output data are in the Optimization DB outside the GRID network.

IV. Conclusion

In this paper, we have presented an experimental framework that allows the storage of all data involved in optimization algorithms for JSS problem and execute

them in a distributed enviroment called the GRID. This system can overcome problems arising in optimization studies requiring computational power and repetition of same experiments.

We propose the use of two databases: Optimization DB and Analyze DB. The first database has the objective of collecting and feeding the execution of the experiments maintaining a fast update and consistence data. The second database has the objective to analyse the data in a fast way and maintaing the assumptions made for the analyse without updates in real time.

This framework is design for the JSS problem, however some modification can be done to support any optimization problem like adding a more general way to store the problem instances.

The framework can be used to provide the scientific community an infrastructure for collecting data, e.g.:

- Benchmark new algorithms with existing ones
- Confirm results published in scientific articles
- Avoid the repeated execution of experiments

References

[1] Pedro Abreu, Carlos Soares, and Jorge M. S. Valente. Selection of heuristics for the job-shop scheduling problem based on the prediction of gaps in machines. In *LION*, pages 134–147, 2009.

[2] Thomas Connolly Carolyn Begg. *Database Systems: A Practical Approach to Design, Implementation and Management*. Addison-Wesley, 2004.

[3] A. S. Jain and S. Meeran. Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research*, 113:390–434, 1999.

[4] Dirk C. Mattfeld, Cjristian Bierwirth, and Herbert Kopfer. A search space analysis of the job shop scheduling problem. *Annals of Operations Research*, 86:441–453, 1999.

[5] JanN. Rijn, Bernd Bischl, Luis Torgo, Bo Gao, Venkatesh Umaashankar, Simon Fischer, Patrick Winter, Bernd Wiswedel, MichaelR. Berthold, and Joaquin Vanschoren. Openml: A collaborative science platform. In Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Železný, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 8190 of *Lecture Notes in Computer Science*, pages 645–649. Springer Berlin Heidelberg, 2013.

[6] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285, 1993.

[7] Jakob Vesterstrom and Rene Thomsen. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 2, pages 1980–1987. IEEE, 2004.