University of Amsterdam

MSc System and Network Engineering
Research Project 1

Framework for profiling critical path related algorithms

by
Henri Trenquier
11758929

February 21, 2018

6 ECTS
January, 8^{th} – $February, 12^{th}$

Supervisors: Dr Zhiming Zhao,
Dr Arie Taal

Abstract

Critical path related algorithms are used to solve time critical applications and optimises costs for data processing with cloud computing while respecting time constraints. In this paper we will try to create a framework to profile these critical path related algorithms thanks to shared standards and common testing environment. The framework would also aim at gathering the community with new standards to ease work sharing, alleviate paper reviewing and reduce the *reproducibility crisis* [7].

Contents

1 Introduction

Cloud computing has recently emerged and becomes the most popular choice to run complex applications that require large scale data processing. Developing complex distributed cloud applications that need to meet high Quality of Service (QoS) and Experience (QoE) is a challenge. The Software Workbench for Interactive, Time Critical and Highly self-adaptive cloud applications (*SWITCH*[4]) is a European scientific project aiming at developing solutions for this global problem. Planning virtual infrastructure for time critical applications with deadline constraints is an issue tackled by the SWITCH project. This research is lead in the context of this project. The general scheduling problem is known as an NP-hard problem [13]. The complexity arises from the many parameters such as the time and ordering constraints, the proposed performance models or the different workflow topologies.
As a result, many scheduling algorithms are proposed in papers. In particular, a "Cost Effective Deadline Aware" (CEDA) scheduling strategy[10] has been proposed. We can cite the *IC-PCP* algorithm as an example. But these algorithms, as well as cloud computing in general, do not show deterministic performance behaviours. The community needs to categorize the workflow topologies and rate the algorithm's performance depending on the different topologies. Moreover, as research papers only show successful results, and often only present pseudo codes, it is harder to determine the quality of the algorithm.

2 Related Work

Use of workflows to solve large scale data processing is frequent. The *Pegasus*[3] project has created a workflow management system that executes these workflow-based applications [2].

Workflows are a means to express the dependencies between the multiple tasks in complex applications. Most of the time, workflows are described by acyclic directed graphs.

2.1 Graphs and DAGS

DAGs and also graphs in general are a very powerful data structure. They can represent various kinds of objects such as molecules but also model scientific problems. In this case, they represent data processing workflows and allow us to have a clear idea of the different tasks and their dependencies. Since graphs have many use-cases, many researches have tried to classify them. This motivation hails from the complexity of the problems that can be modelled with this data structure. The behaviour of an algorithm depends on the topologies, number of nodes, mean in-degree and many other (un)known metrics that can be defined on a graph. A research has tried to introduce a new metric to define a hierarchy of classes of DAGs [8]. Other researches have tried to classify graphs by mining patterns with *graph boosting methods* [12] or *kernel methods*[9]. Classification of DAGs has not been attempted yet.

2.2 State of the art

The following research paper lists the most popular workflow scheduling algorithms that have been used in Grid projects [14]. A recent paper[6] written in 2013 introduces a new algorithm which focuses on minimizing the cost of a workflow in an Infrastructure as a Service. This algorithm is called IC-PCP which stands for IaaS Cloud Partial Critical Path. The 300+ citations speak for its popularity. This algorithm seems to show very promising results. However, the paper formulates some confusing phrases in the pseudo-code that can be interpreted in different ways. Thereby, the results are hard to reproduce since the authors did not share their version of the algorithm's code.

3 Research Question

In order to test and compare Critical path related algorithms and tackle the reproducibility crisis, we define the research question as follows:

Can we profile Critical path related algorithms by the means of a framework ?

We can divide this question into two sub-questions:

- How relevant is a framework for profiling critical path related algorithms ?
- What are the challenges for such a framework ?

4 Prerequisites

Before diving into the details of the prototype, we will, in this section, introduce some prerequisites. An example will be introduced in the end of this section for a good understanding of the problem.

4.1 Workflow

A workflow is a sequence of sub-tasks that describes a big and complex project or process. Such a process needs to be depicted to be able to share the work between different people or resources.

4.2 Directed Acyclic Graph

Directed Acyclic Graphs or *DAG* is the kind of graph used to represent workflows.

Definition [5]

A directed graph $G = (V, E)$ consists of a nonempty set of nodes V and a set of directed edges E. Each edge e of E is specified by an ordered pair of vertices $u, v \in V$. A directed graph is simple if it has no loops (that is, edges of the form u -> u) and no multiple edges.

A directed acyclic graph is a directed graph that has no cycles ie. if there is no path from u to u, $u \in V$.

Model

In this model, V represents the set of tasks to be executed. E represents the dependencies between the different tasks. An edge, (or ordered pair of vertices (u, v)) represents the order constraint: v can only be executed if u is finished. A vertex that has no predecessor is an *entry node* and a vertex that has no successor is an *exit node*.

If there are more than one entry and/or exit node, we usually introduce a dummy entry and/or exit node. The virtual node will have his computing and communication costs equal to zero. For practical reasons, these two additional nodes are often added to help the algorithm and have no consequence on the final result.

In the next pages, we will assume that the entry and exit nodes always exist and are unique.

4.3 Deadline

For each generated workflow, the user defines a deadline. We define the Deadline as the time limit to execute the workflow. In this project, we will vary the deadline in order to stress more or less the scheduling algorithm (see Subsection **4.5 Scheduling algorithms**.

4.4 Critical Path

The critical path is the longest series of sequential operations in a worflow representation, if the workflow is represented by a DAG, i.e. the longest path from entry node to exit node.

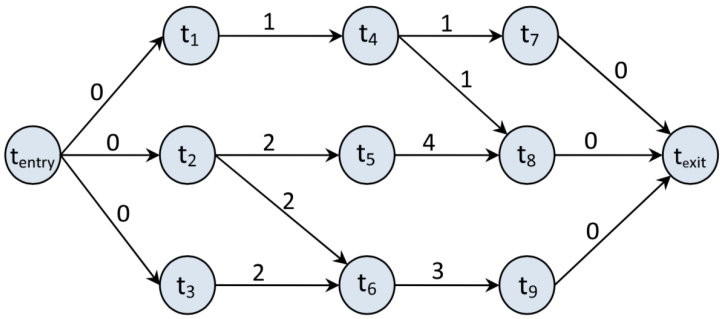


Figure 1: Sample DAG

An example of critical path of the sample DAG Figure 1 can be computed thanks to the tasks' costs given by performance model (Table 1). The critical path is $\{t_2, t_6, t_9\}$.

The time of the critical path defines the shortest deadline applicable for this work The shortest deadline that can be validated by an algorithm is greater or equal than an sum of the critical path nodes computation time.

Further in the paper, we will introduce a parameter to measure the performance of algorithm which relies on this minimum deadline value.

4.5 Performance model

The performance model describes the available computation services and their execution time for each task. Different models can be generated for the same graph. If we take the sample performance model described by the Table 1 from the IC-PCP sample [6] again.

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
S_1	2	5	3	4	3	4	5	3	5
S_2	5	12	5	6	8	8	8	6	8
S_3	8	16	9	10	11	11	11	8	14

Table 1: Performance model

4.6 Scheduling algorithms

Scheduling algorithms are designed to map tasks from the workflow to Virtual Machines (VM). As described in the *Directed Acyclic Graph* subsection, the tasks to map are represented by the nodes of the DAG. Also, the VMs' quality are described by the performance model.

- The performance model describes the time that each individual task of the workflow will take to compute on the VM available in a cloud. An example is given in Table 1.
- Each type of VM has a different price.
- At the end of the run, the algorithm outputs a configuration corresponding to the mapping of all tasks VMs. An example is also given in Table 3.

It is possible to assign all tasks to the most powerful type of VM. However, this VM will also be the most expensive, making the workflow's execution very expensive too. Hence, the principle of the scheduling algorithms is to reduce the cost of the workflow's execution by mapping cheaper VMs to the tasks. In order to have a valid solution, the workflow's execution time has to respect time constraints: the deadline.
Finally, a scheduling algorithm takes as input:

- a DAG
- a deadline
- Communication costs (stored as the DAG edges' weights)
- Computation costs (stored in the performance model)

- VMs prices

We will call the output of a scheduling algorithm the *final configuration.*

Defining a shorter deadline will stress more the algorithm. We will prefer an algorithm that is able to solve problem with short deadlines.

The scheduling algorithms that we will test and profile use the critical path to do the mapping. We call these algorithms "Critical path related algorithms".

5 Framework

5.1 Motivations

The motivation to create a framework for profiling critical path related algorithms is dual.
On the first hand, cloud computing is frequently a viable option for a company or a research project to run distributed applications. Cloud providers work with a "pay-as-you-go" business model. Hence, depending on the management of the resources, the operating expenses can vary a lot. Consequently, a company will be more inclined to invest in a good scheduling algorithm that will lower the costs of the service.
Researchers have come up with multiple scheduling algorithms. But it is difficult to compare their performances and to know with what kind of workflow the algorithm performs better.
The framework aims at proposing common input datasets and analyse the results. Based on the same Key Performance Indicators we could rank the tested algorithms. On the other hand, researchers usually have their own testing environment, algorithms with different input formats etc. This heterogeneity of environments makes it harder to share the work and reproduce the results. The idea of a common framework would foster the researchers to work on common standards. It could be also a solution for the *Reproducibility Crisis* raised by Monya Baker in a fairly recent survey[7].

5.2 Prototype

In order to grasp the challenges of building this profiling tool, we have developed a prototype. The next pages of this paper will describe the issues tackled during the research to allow the reader to understand the different challenges.

To fit in the time constraints of the research we have defined a reasonable scope:

- The prototype will compare 3 different implementations of the IC-PCP algorithm (these implementations have been written by two different people)
- Every code will be tested on the same datasets (the datasets will be exposed in section **5.5 DAG datasets**)
- Key Performance Indicators:
  - Computation time
  - Percentage (cf. Formula 1)
  - Success ratio (cf. Formula 2)

  They will be defined in the next subsection.

5.3 Key Performance Indicators

The ranking of the different implementations will be based on the Key Performance Indicators (KPI). The choice of these KPIs is essantial for profiling algorithms.

5.3.1 Computation time

To measure the computation time, we used the time function in the library of the same name. The time is saved before and after the algorithm's code. The result is the difference of the two timestamps.

5.3.2 Percentage

As stated by the definition of the Critical path, a workflow has a minimum deadline. This minimum is equal to the time of the critical path. The percentage p will be defined as follows:

$$deadline = \frac{100 * critical\_path\_time}{p} \tag{1}$$

Indeed, if $p = 100$, the deadline will be as long as the critical path time. This is the most stressful situation for the algorithm.
On the other hand, for $p = 0$, the deadline is infinite. We will discard this value of p. For a $p = 50$, the deadline is twice as long as the critical path time.

So the value of p will vary between 0 and 100, 0 excluded: $p \in (0; 100]$.

5.3.3 Success ratio

For a specific dataset of N DAGs, we define the success ratio of the algorithm:

$$SuccessRate = \frac{NumberOfValidConfigurations}{NumberOfDAGsTested} \quad (2)$$

5.4 Architecture

The prototype is a python script. The script is composed of three main functions.

1. *main.py*
2. *run.py*
3. *plot.py*

A global overview is given in Figure 2. To simplify, the *run.py* function is not represented here. It can be just part of the *main.py* function.

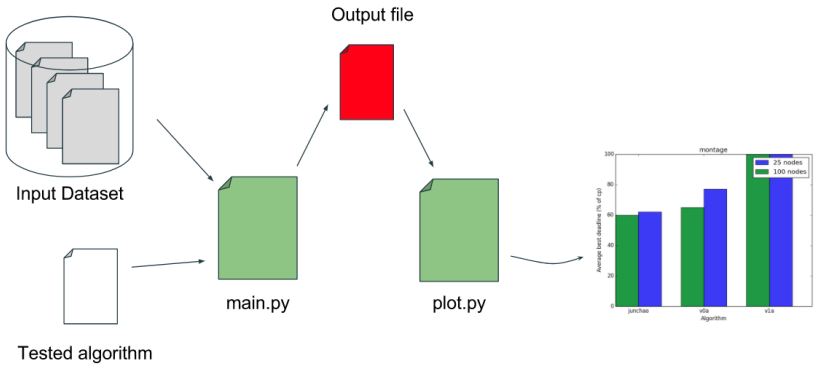


Figure 2: Prototype Architecture

In this subsection, we will see in further details the three main functions.

5.4.1 main

This function is capable of walking the DAG database, and choosing the input dataset depending on the type of topologies.

5.4.2 run

Given an algorithm A and a DAG file D, the *run* function will call the algorithm's main function for a range of values of input parameter p. Here, the percentage will vary from 1 to 100.

Two sub-functions can be called:

- *run_full_range*
  For all integer values of p between 1 and 100, the algorithm A is ran on the DAG D with the deadline computed from p. The success (or validity of the configuration given by the algorithm), the time to compute the configuration, the DAG's name and the percentage for the configuration are saved in a file.

- *run_dichotomy*
  This function only tests the algorithm for specific values of p. It aims at finding the shortest deadline that the algorithm is able to validate. In order to lower the number of runs for time complexity issues, *run_dichotomy* looks for the shortest deadline that the algorithm is capable of reaching. This is done by trying several deadlines based on a dichotomy pattern.

  This 'run method' is based on the following assumption:

  "If the algorithm A succeeds for a shorter deadline then it will also succeed for a longer deadline."

  In other words, we can find a unique p above which the algorithm will not be able to find a valid configuration and under which all configuration is valid (we remind the reader that, according to the Formula 1, the higher the p, the shorter the deadline). However, this assumption, even though it seems natural, will be proved to be wrong.

  Both methods output a file. This file will later be read by the *plot* function. In this output file, each line correspond to a run. The following example shows two lines of an output file.

```
1.30.5;3;0;[22, 23, 24, 27, 29, 30];0.0295469760895
1.30.5;4;0;[22, 23, 24, 27, 29, 30];0.0297148227692
```

  The data is comma-separated values and written in the following order:

  1. DAG name
  2. Percentage
  3. Success or failure (respectively 0 or -1)
  4. Critical Path
  5. Computation time

5.4.3 plot

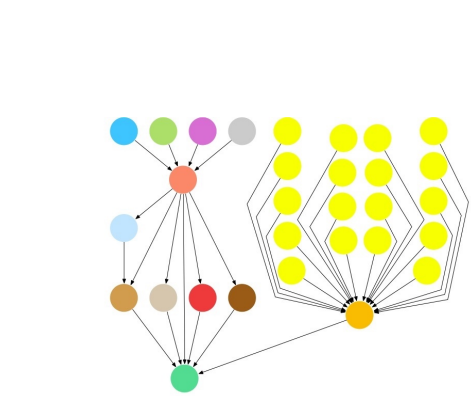
The *plot* function has several interesting methods.

- *getMetadata*: output file names are specific. They contain the input DAG dataset and the name of the algorithm that has been tested. *getMetadata* retrieves the metadata from this file.
- *verifyInputDataset*: this function checks if the DAGs that have been tested are the same. Indeed, to plot and compare the performance results, it is important to have the same input dataset.
- *plot**: There are two plot methods. Each method plots a different performance graph. Performance graphs will be shown later. It is possible to create more methods to plot other interesting relations.

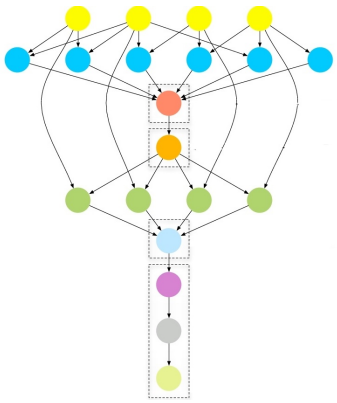
5.5 DAG datasets

The DAG datasets have been generated from sample topologies available on the *Pegasus* [3] website. We call a type of DAG a "topology" since it is the only way to classify them for now. The full database is composed of 5 different topologies. These topologies are used in scientific projects and are available on the pegasus website[3].
The 5 types are:

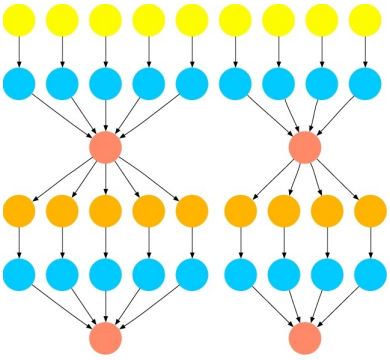
- Sipht : Figure 3(a)
- Montage : Figure 3(b)
- Inspiral : Figure 3(c)
- Epigenomics : Figure 3(d)
- CyberShake : Figure 3(e)



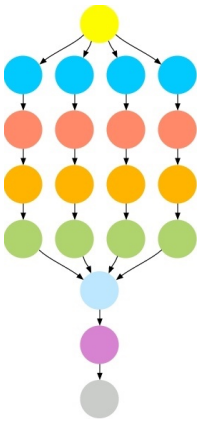
(a) Sipht



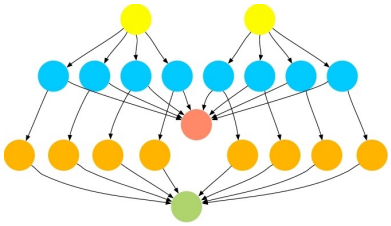
(b) Montage



(c) Inspiral



(d) Epigenomics



(e) Cybershake

Figure 3: Topologies used to generate the datasets

For each topology we have generated multiple instances by changing the weights of edges and the performance model.

It is possible to generate DAGs randomly, but we prefer to test algorithms on DAGs used in practice. Moreover, these topologies have already been used in other critical path related papers [6].

Input format
There are three input files for a DAG for this prototype.

- *1.25.1.propfile* contains the actual topology of the graph.
- *performance* contains the performance model.
- *price* contains the price of each VM described in the performance model.
- *deadline* contains a deadline information (only used if a percentage has not been set).

5.6 Example

This subsection will recapitulate the previous explanations on an example taken from the IC-PCP paper[6]:

- The graph Figure 1 shows the sample DAG.
- The following code "sample.propfile" describes the DAG Figure 1.

```
    digraph dag {
1 -> 4 [weight=1.0];
2 -> 5 [weight=2.0];
2 -> 6 [weight=2.0];
3 -> 6  [weight=2.0];
4 -> 7  [weight=1.0];
4 -> 8  [weight=1.0];
5 -> 8  [weight=4.0];
6 -> 9  [weight=3.0];
}
```

  We can see that the entry and exit nodes are not in the *propfile*. These nodes are automatically added in the algorithm.

- The performance model describes the available computation services and their execution time for each task. Any performance model can be applied to this graph. If we take the sample performance model described by the Table 1 from the IC-PCP sample [6] again.

  The following code describes the performance model described by Table 1

```
2 5 3 4 3 4 5 3 5
5 12 5 6 8 8 8 6 8
8 16 9 10 11 11 11 8 14
```

VM	Price
S_1	5
S_2	2
S_3	1

Table 2: Prices of VMs

- VM prices are chosen by the service provider. The prices given by the Table 2 are described in a file called "price".

  The code for this pricing model is given as follows:

```
5 2 1
```

- The deadline is simply a file with the deadline value in it (e.g. 291)
- The final configuration is the distribution of the workflow's tasks on the VM instances.

  It is presented as the Table 3

	Start time	Stop time	Duration	Total cost	Assigned tasks
$S_{2,1}$	0	28	28	6	t_2, t_6, t_9
$S_{2,2}$	14	28	14	4	t_5, t_8
$S_{3,1}$	0	9	9	1	t_3
$S_{3,2}$	0	29	29	3	t_1, t_4, t_7

Table 3: Final configuration for the sample DAG

6 Experiments

On all experiments, we compare three implementations of the same algorithm (*v0*, *v1* and *v2*).

6.1 Experiment 1

The first experiment uses the method *run_ dichotomy*.

The result of the first experiment is displayed on the graph figure 4. The dataset used for this was generated from the Montage topology.

As explained in the **5.4.2 *run*** subsection, we run the tested algorithm (*v0*,*v1* or *v2*) on a DAG and its performance model taken from the dataset. The function finds the highest percentage (ie. shortest deadline) that the algorithm is able to respect.

For each run we have output the highest percentage.
Green bars show the mean of the highest percentage for all DAGs of 25 nodes

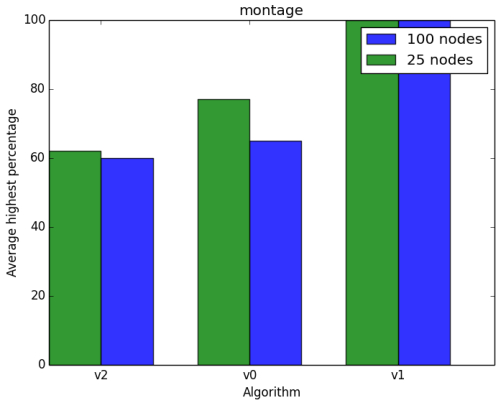


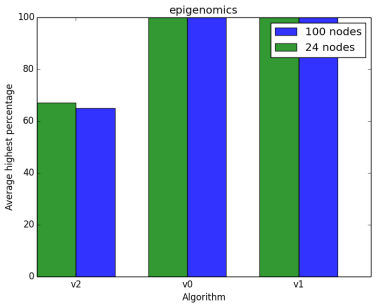
Figure 4: Experiment 1 : Performance on Montage

in the dataset. Blue bars also show the mean of the highest percentage but for DAGs of 100 nodes. On the y axis, is the mean of the highest percentage. Hence, a high bar means that the algorithm can find a configuration in a very short deadline.

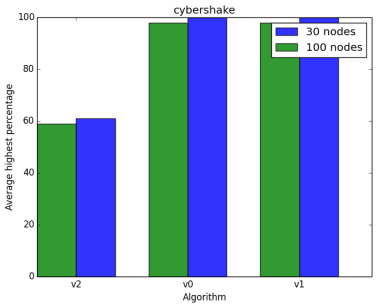
The most interesting results are obtained with the Montage topology. Compared to all other tested topologies, Montage seems to be the hardest to solve (at least for algorithm *v0*).

From the other performance graph, we can see that the number of nodes has a low impact on the average best deadline.

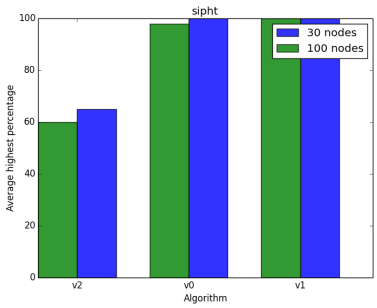
Figure 5 shows the results for the 4 other topologies. The results are quite similar to each others:



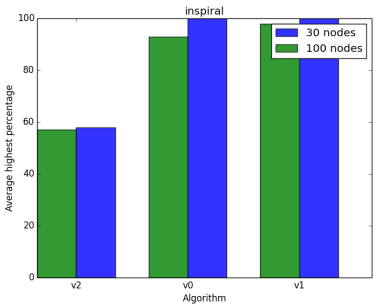
(a) Epigenomics



(b) Cybershake



(c) Sipht



(d) Inspiral

Figure 5: Exp 1: Results for Epigenomics, Sipht, CyberShake and Inspiral

6.2 Experiment 2

The second experiment will rank the implementations based on the computation time for a run and success rate.

On the shared x axis, is the percentage: closer to 100 means shorter deadline, $x \in (0; 100]$.
The top graph shows the success rate (in %) , how many run succeeded versus the percentage. The bottom graph shows the Computation time in second. The computation has been performed for all integer values of p between 1 and 100 this time.

In the same way as the first experiment, we will show the results for the Montage dataset : Figure 6.

Before the experiment, we did not expected the success rate to increase at all. However, Figure 6 clearly shows that all algorithm's successes drop around p=75% (ie. when the deadline is $1.33 * critical\_path\_time$) but raise after.

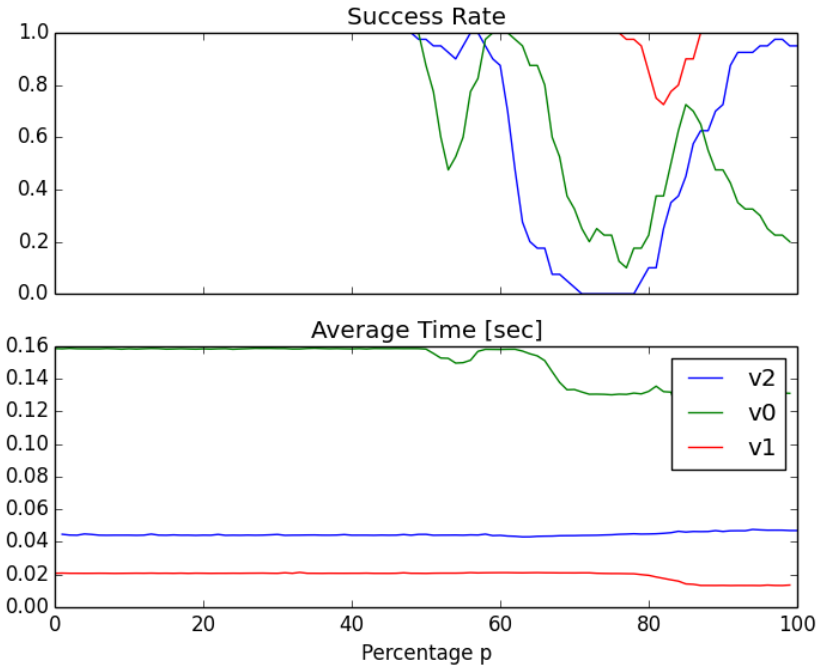


Figure 6: Experiment 2 : Performance on Montage

This mean that the assumption stated in subsection **5.4.2** is not true:

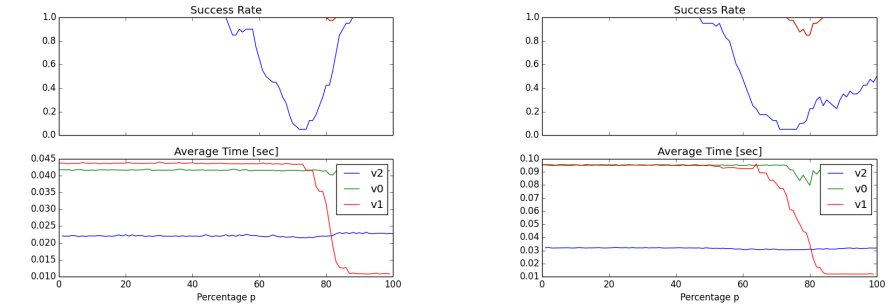
Sometimes, an algorithm can fail for a longer deadline and succeed for a shorter deadline.

This statement also seem to be true for all 3 implementations on Montage since we can see a little gap on the top diagram.

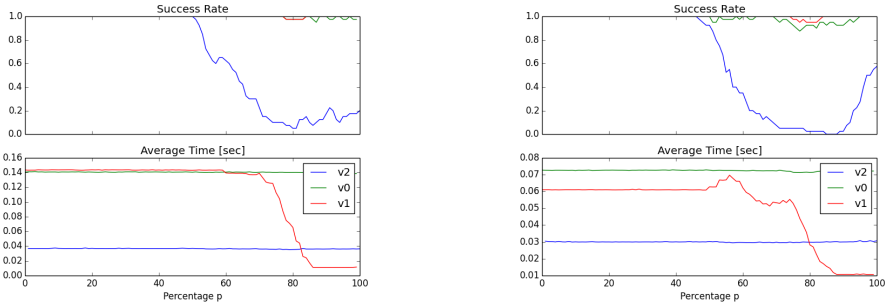
On the other hand, we can see that the algorithm **v1** shows better performance overall on all KPIs:

- Shortest deadline (Experiment 1)
- Computation Time (Experiment 2)
- Success Rate (Experiment 2)

On Figure 7, we can see that the result are also quite similar to each others. In the reading order, we can see the results of Epigenomics, Sipht, CyberShake and Inspiral for Experiment 2.



(a) Epigenomics (b) Cybershake



(c) Sipht (d) Inspiral

Figure 7: Exp 2: Results for Epigenomics, Sipht, CyberShake and Inspiral

7 Discussion

7.1 Experiment 1

The first experiment is based on the assumption stated in paragraph **5.4.2**. As demonstrated by the second experiment, this assumption is wrong. Consequently, the results shown in the first experiment are wrong too. We have purposely kept these results in the paper for 2 reasons.

- The results obtained with this experiment are very similar to the results of the second experiment. Thus, we can state that they reflect the reality even though they are based on a false statement.
- We wanted to show that we have been able to gainsay our own statement. Thereby, we have shown that the diversity of experiments is able to teach us new knowledge on the behaviour of these algorithms.

7.2 Experiment 2

As stated in the previous subsection, this second experiment allowed us to refute the assumption. As we had stated in the first experiment, the number

of nodes did not show a great impact on the successes. However, the second experiment's dataset does not distinguish the DAGs with their number of nodes. It could be interesting to separate results depending on the number of nodes for the computation time.

7.3 Prototype

The prototype proposed in this project is used as a proof of concept. As it has been developped in a limited period, it has a lot of limitations. Among them, the most important one are the following:

- For each implementation of an algorithm to be tested, it is necessary to adapt its input files formats. The prototype has been developped around the algorithms *v0* and *v1*. However, the algorithm *v2* did not take in charge the ".propfile" format. A parsing method had to be added before to enable the algorithm to read from the ".propfile" format.
- For a run with the *run_full_range* function, an output file is generated after $100 * 40$ runs (100 values of the percentage and 40 different DAG instances). This large amount of run needs fast computing resource or will take a considerable amount of time.
- For this prototype, the solution validity check is coded in the algorithm implementation. That should be done in the framework.
- The output file is not optimised. We have chosen to write the critical path of the DAG in the output file but we did not use it for plotting results. Moreover, it would have been more interesting to output the cost of the final configuration and the configuration. These are 2 important pieces of information that are important for integrity of the results and as KPI.
- The DAG database is short. The number of algorithms to profile is low and the database only has 200 DAGs, 40 per topology.
- The percentage metric defined in this project can be confusing in the beginning. It may be interesting to define a more intuitive one.

The framework will make even more sense when we have a lot of values to compare, and also more experiments.

8 Conclusion

The research question was:

Can we profile Critical path related algorithms by the means of a framework ?

We can now answer this question positively.
Even though we did not have a large amount of algorithms to check nor a big DAG database, we have actually been able to define a ranking regarding three different key performance indicators.

Moreover, we have been able to detect an interesting tendency of one of the five topologies: Montage. The refutation of the assumption is also an unexpected advantage of the framework.

At the end of the day, the framework has shown interesting results with few means. The real potential of this framework now relies on the community's implication. This is what we will describe in the next and final section of this report.

The framework will be available on the following source [1].

9 Future Work: Final Framework

Ideally, we would like the framework to answer all question of this kind:

What algorithm is the best for a topology like Inspiral ?

Indeed, as the scheduling problem is known as NP-hard[13], we don't expect to find a universal "best" algorithm for all topologies any time soon. However, it is possible to greatly enhance and ease the work of the researchers and developers for this scheduling problem.

During this project, we have been confronted with many obstacles. The following features are the challenges that we have been able to identify.

- **Universal standards**
  As we had issues adapting each implementation to be usable by the framework, it is necessary to establish universal standards. These standards, such as input DAG file format or input parameters such as the percentage in the algorithm itself.
- **Web-application**
  It is important if we want the community to adhere to this project to make the framework highly available. A web-application would be the best solution. Nothing else than a internet browser would be needed.

- **Community databases**
  Members of the community should be able to upload their work and share it. We should be able to find in this database implementations of algorithms as well as new topologies. An uploaded algorithm should be available to other users to satisfy the open source value of the SWITCH project [4].

- **Percentage**
  The percentage metric that has been defined in this project is confusing. Moreover, a value of p e to 1% might not be a challenging case for an algorithm. In the future, when the algorithms will show very good results, it will be more interesting to focus on the values above 50%. Thus, this metric could be enhanced.

- **Performance**
  Performance has already been a little issue in this project. Computing tests on the whole DAG database takes time. If many users want results quickly from their new algorithm, it is important to make this application scalable.
  The first step towards a faster back end application would be to make the *run* script mutli-threaded. In addition, powerful libraries such as graph-tool[11] could be used to manipulate graphs.

- **New features**
  Furthermore, new features should be added on the framework. These features are listed Table 4.

Feature	Description
DAG Generator	Create and visualize new topologies
DAG Parsing	Manage different input topologies format
input checker	Check input algorithm parameters
Solution checker	Make sure the final solution is valid

Table 4: Framework features

The solution checker is very important for the integrity of the results on the framework. This feature should be developed so that users do not check their own algorithms.
This part of the framework should be very secure as well as the output files integrity. In order to be a trusted platform, this data should not be tampered with. This would poison the framework and its users.

If this requirement is met, the framework would ease the work of reviewers and greatly help with reproducibility by providing an authoritative source for researchers.

References

[1] Critical path related algorithm profiling framework. Link: `https://github.com/htrenquier/cp-related-algorithm-profiling-framework.git`.

[2] Ligo project. Link: `https://www.ligo.caltech.edu/`.

[3] Pegasus. Link: `http://pegasus.isi.edu/`.

[4] Switch project. Link: `http://www.switchproject.eu`.

[5] Directed graphs. *mcs-ftl*, page 189, 2010.

[6] S. Abrishami, M. Naghibzadeh, and D. H. Epema. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future Generation Computer Systems*, 29(1):158 – 169, 2013. Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures.

[7] M. Baker. 1,500 scientists lift the lid on reproducibility. *Nature News, Springer Nature*, 2016.

[8] M. Chaturvedi. A parametric classification of directed acyclic graphs. 2017.

[9] K. T. H. Kashima and A. Inokuchi. Marginalized kernels between labeled graphs. *In Proceedings of the 21st International Conference on Machine Learning*, page 321 – 328, 2003.

[10] R. A. Haidri, C. P. Katti, and P. C. Saxena. Cost effective deadline aware scheduling strategy for workflow applications on virtual machines in cloud computing. *Journal of King Saud University - Computer and Information Sciences*, 2017.

[11] T. P. Peixoto. The graph-tool python library. *figshare*, 2014.

[12] H. Saigo, S. Nowozin, T. Kadowaki, T. Kudo, and K. Tsuda. gboost: a mathematical programming approach to graph classification and regression. *Machine Learning*, 75(1):69–89, Apr 2009.

[13] J. Ullman. Np-complete scheduling problems. *Journal of Computer and System Sciences*, 10(3):384 – 393, 1975.

[14] J. Yu, R. Buyya, and K. Ramamohanarao. *Workflow Scheduling Algorithms for Grid Computing*, pages 173–214. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.