Cloud Workload Prediction by Means of Simulations

Gabor Kecskemeti
Liverpool John Moores University
Department of Computer Science
Liverpool, United Kingdom
g.kecskemeti@ljmu.ac.uk

Attila Kertesz
University of Szeged
Szeged, Hungary
keratt@inf.u-szeged.hu

Zsolt Nemeth
Institute for Computer Science and Control (MTA SZTAKI)
Budapest, Hungary
nemeth.zsolt@sztaki.mta.hu

ABSTRACT

Clouds hide the complexity of maintaining a physical infrastructure with a disadvantage: they also hide their internal workings. Should users need to know about these details e.g., to increase the reliability or performance of their applications, they would need to detect slight behavioural changes in the underlying system. Existing solutions for such purposes offer limited capabilities. This paper proposes a technique for predicting background workload by means of simulations that are providing knowledge of the underlying clouds to support activities like cloud orchestration or workflow enactment. We propose these predictions to select more suitable execution environments for scientific workflows. We validate the proposed prediction approach with a biochemical application.

CCS CONCEPTS

•Computer systems organization →Cloud computing;

ACM Reference format:
Gabor Kecskemeti, Attila Kertesz, and Zsolt Nemeth. 2017. Cloud Workload Prediction by Means of Simulations. In *Proceedings of CF'17, Siena, Italy, May 15-17, 2017,* 4 pages.
DOI: http://dx.doi.org/10.1145/3075564.3075589

1 INTRODUCTION

Infrastructure as a Service (IaaS) clouds became the foundations of computation/data intensive applications. While IaaSs offer means to control a virtual ensemble of resources (i.e., virtual infrastructures), they provide no means for insight into the state, load, performance of their physical resources. Due to the multi-tenant environment of clouds application performance may be significantly affected by other, (from the point of view of a particular user) unknown and invisible processes, the so-called background workload.

This paper aims at studying performance issues related to the – unknown – background load and proposes a methodology for its estimation. We envision a scenario where modifications in the virtual infrastructure are needed at runtime and to take right actions the background load cannot be omitted. We made two assumptions on the application: (i) it runs long enough so that virtual infrastructure re-arrangement time is negligible and (ii) it is executed repeatedly. Scientific workflows are good candidates for exemplifying such applications. Efficient execution of workflows requires a precise scheduling of tasks and resources which furthermore, requires both timely information on the resources and the ability to control them.

The recurring nature of workflows enables the extraction of performance data. Our work stems from the assumption that by extracting information from *past* workflow executions, one could *identify current* and *predict future* background workloads of the resources allocated for the workflow. The result of this prediction subsequently enables to steer workflow enactment. Our workload prediction aims at finding historic traces, that likely resemble the background of workload behind a currently running workflow.

The paper's main contributions are: (i) the concept of a load prediction method based on the combination of historic traces, (ii) an algorithm for realising the load prediction at runtime so that performance constraints are observed, and (iii) a validation of this approach using a biochemical application with a state of the art simulator using historic traces from a widely used archive.

The remainder of this paper is as follows: Section 2 presents related work, then Section 3 introduces our terminology and assumptions. Section 4 shows our algorithm. Section 5 presents its validation. Finally, the contributions are summarised in Section 6.

2 RELATED WORK

In this paper, we examine traces of the past of certain workflows, and perform a prediction of the expected background load of the clouds. Our technique fits in the autonomous loop of monitor-analyse-plan-execute [1], where we focus on the analyse phase.

Calheiros et al. [2] offers cloud workload prediction using autoregressive integrated moving average. Their model's accuracy is evaluated by predicting future workloads of request traces to web servers. Also, Magalhaes et al. [3] developed a workload model for the CloudSim simulator using generalised extreme value/lambda distributions. They argue that user behaviour must be considered in workload modelling. Our approach share this view: we apply a runtime behaviour analysis to find a workflow enactment plan that best matches the infrastructure load including user activities.

Caron et al. [4] used workload prediction based on identifying similar past occurrences of the current short-term workload history for efficient resource scaling. It uses real-world traces from clouds and grids to identify similar usage patterns to a current window of records, and their algorithm predicts the system usage by extrapolating beyond the identified patterns. In contrast, our work focuses on scientific workflows allowing the analysis and prediction of recently observed execution time discrepancies by introducing simulations to the prediction and validation phases.

The research leading to these results has received funding from the European COST programme under Action identifier IC1304 (ACROSS), and it was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

3 BACKGROUND

An *enactment plan* describes the jobs of a scientific workflow, their schedule to resources and it is processed by a workflow enactor that assigns jobs to resources. In our vision, the enactment plan also lists the projected execution time of each job. Workflow enactors are expected to base the projected execution time on historic executions to represent their expectations according to the job execution speed.

The virtual infrastructures created by the enactor are hosted at IaaSs that tend to feature multi tenancy and under provisioning for optimal costs and resource utilisation. These practices, especially under provisioning, could perturb and potentially hinder the virtual infrastructure's performance. Thus, the workflow enactor should continuously monitor the behaviour of its workflows running on the virtual infrastructure. In case of deviations, actions in the management of the virtual infrastructure should take place.

In this paper, workflows $W \in \mathcal{W}$ (where $\mathcal{W}$ is the set of all possible abstract workflows) are represented as an ordered set of jobs: $W = \{j_1 \dots j_N\}$, where the total number of jobs in the workflow is $N \in \mathbb{N}$. The job order is set by their projected completion time on the virtual infrastructure whereas the job dependencies are kept in the domain of the workflow enactors. The projected execution time of the a job ($j_x \in W$) is $r_{ex}(j_x)$ – where $r_{ex} : W \to \mathbb{R}^+$.

We refer to a workflow instance (i.e., an execution of the abstract workflow W) with the couple: $[W, t] : \mathcal{W} \times \mathcal{T}$ – i.e., the workflow and the start time (t) of its first job $[j_1, t]$. Hence, all instances of $j_x \in W$ are also identified as $[j_x, \mathcal{T}] : j_x \in [W, \mathcal{T}]$. Once the workflow started, the enactor's monitoring facilities will collect the observed execution times for each job instance. We denote these as: $r_{ob}(j_x, t)$ – where $r_{ob} : W \times \mathcal{T} \to \mathbb{R}^+$.

Using the acquired data from the monitoring facilities, we define our error function of (partial) *workflow execution time* to determine the deviation in execution time a workflow suffered compared to its enactment plan. It is partial as the workflow instance is split into two parts: jobs $j_1, \dots j_k$ already executed whereas $j_{k+1}, \dots j_N$ are not yet complete. Hence, ($E : \mathcal{W} \times \mathcal{T} \times \mathbb{N} \to \mathbb{R}^+$):

$$E(W, t, k) := \sqrt{\frac{\sum_{1 \le i \le k}(r_{ex}(j_i) - r_{ob}(j_i, t))^2}{k}} \tag{1}$$

where $\forall j_i \in W$. The higher the error value the higher the deviation of the instance from its enactment plan (r_{ex}). The enactment plan likely contains projected values that are carefully selected by the enactor to meet the needs of the workflow's user and follow the capabilities of the used cloud resources. Thus, our error function indicates if the fulfilment of the projected values are at risk.

The deviation from the projected execution times (as indicated by the error function) could either be caused by (i) an unforeseen reaction to a specific input or by (ii) the background load behind the virtual infrastructure of the workflow. Deviations of case (i) are rare, because job execution times usually follow a Pareto distribution [5]. The long execution times in the slowest 5 % of the jobs cannot be caused by background load. On the other hand case (ii), under-provisioning is frequent in cloud environments and can cause background load variation yielding observable (but minor) perturbations in job execution times. In this paper, we focus on case (ii) only. Consequently, when observing a significant increase in job execution time compared to the projected one, we assume case (i) and we do not apply our technique.

4 WORKLOAD PREDICTION

Workload prediction is expected to be initiated after a job (j_k) completes, if the error function shows significant deviations: $E(W, t, k) > E_\epsilon$, where E_ϵ is predefined by the workflow developer. The maximum time spent on workload prediction is limited by a predefined $\mathfrak{T}$ (also set by the workflow developer). Despite deviations, workload prediction is not performed if $\sum_{i=k+1}^{N} r_{ex}(j_i) < \mathfrak{T}$.

We apply a practical approach: we simulate the workflow execution (according to the enactment plan) on a given cloud infrastructure while adding *known* workloads as background load. We expect that observed execution times in the simulation would closely match their real-world counterparts if the added background load also closely estimates the real-world load. Before the details, in the next paragraphs, we provide a few definitions.

A *trace fragment* is a list of activities characterised by such runtime properties (e.g., start time, duration, performance, etc.) that are usable in simulators. Each fragment represents realistic workloads i.e., real-world behaviour. Fragments are expected to last for the duration of all past ($j_i, 1 \le i \le k$) and future ($j_i, k+1 \le i \le N$) jobs, in a worst case serial execution $\sum_{i=1\dots k} r_{ob}(j_i) + \sum_{i=k+1\dots N} r_{ex}(j_i)$. Apart from their length, fragments are identified by their starting timestamp denoted as $t \in T$ (where $T \subset \mathcal{T}$). Later we refer to particular fragments by their starting timestamp.

Arbitrary selection of fragment boundaries would result in millions of trace fragments. Their analysis would take days for each workflow instance. However, workflow developers typically allow only a few minutes for prediction – $\mathfrak{T}$. To reduce analysis time, fragments need to be *pre-filter*ed so only a few of them ($T_{filt} \subset T$) are used. Pre-filtering can use approaches like pattern matching, runtime behaviour distance minimisation (e.g., by comparing to past stored workflow behaviour), or even random selection. These techniques are out of scope of this paper.

Alongside fragments, several error values are also collected and stored in relation to the past workflow instances. Just like projected execution times, these values steer the algorithm. First, as *past errors*, we store the values for every possible k from our previously partial execution time error function (Eq. 1). We also store future errors: we use the part of the workflow containing the jobs after j_k: $W^F(W, k) := \{\forall j_i \in W : i > k \land i \le N\}$, where $W^F \in \mathcal{W}$. Thus, the future error function determines how a previously executed workflow instance continued after a specific past error value:

$$F(W, t, k) := E(W^F(W, k), t, N - k). \tag{2}$$

4.1 Overview of the algorithm

Algorithm 1 searches for a timestamp so that the future estimated error is minimal, while the simulated past error is the closest to the actual past error. We assume that similar error values in past (simulated/real) workloads would result in similar future workloads.

In detail, line 1 randomly picks a starting timestamp (t_{init}) of the fragments from T_{filt} . This fragment will be initial approximation. In line 16, this t_{init} is updated so it better approximates the background load (T_{list} stores the updates). The search window – $\mathfrak{R}$ of line 4 – is a set of timestamps within a $S/2$ radius from t_{init}.

A simulator calculates ($sim(W, R_{ex}, i, t)$) observed execution times r'_{ob} for the jobs in the simulated infrastructure (see line 8). The $E'(W, t, k)$ – error of simulated execution time – function shows

Algorithm 1 Fitting based prediction

```
Require: $S \in \mathbb{R}^+$ – the maximum spread for minimum search
Require: $I \in \mathbb{N}$ – the maximum iteration count
Require: $P \in \mathbb{R}^+$ – max evaluations for searching in function $\phi(x)$
Require: $[W, t_{curr}]$ – the current workflow instance
Require: $R_{ex} := \{r_{ex}(j_i) : \{j_i \in W\}\}$ – the model execution times
Require: $R_{ob} := \{r_{ob}(t_{curr}, j_i) : \{j_i \in W \wedge i \leq k\}\}$ – the observed
    execution times
Ensure: $t_{target}$ is around the approximated workload
 1: $t_{init} \leftarrow t \in T_{filt}$
 2: $T_{list} \leftarrow \emptyset$
 3: repeat
 4:     $\Re \in 2^(t_{init} - S/2, t_{init} + S/2) \setminus \{\emptyset\}$ – arbitrary choice
 5:     for all $t \in \Re$ do
 6:         for all $j_i \in W : i < k$ do
 7:             $r'_{ex}(j_i) \leftarrow r_{ob}(t_{curr}, j_i)$
 8:             $r'_{ob}(j_i) \leftarrow sim(W, R_{ex}, i, t)$
 9:         end for
10:     end for
11:     $\phi(x) := \sum_{t \in \Re} \left|E'(W, t, k) - E(W, x + t - t_{init} + S/2, k)\right|$
12:     $T_{red} \in 2^{(T \setminus T_{list})} : |T_{red}| = P$
13:     $t_{min} \leftarrow \min\{t \in T | \phi(t) = \min_{x \in \{T_{red}\}} \phi(x)\}$
14:     $T_{list} \leftarrow T_{list} \cup \{t_{min}\}$
15:     $t_{target}(|T_{list}|) \leftarrow \{t_l \in T : F(W, t_l, k) = \min_{t_{min} - S/2 < t < t_{min} + S/2} F(W, t, k)\}$
16:     $t_{init} \leftarrow t_x \in T_{filt} : |t_x - t_{target}(|T_{list}|)| = min_{t \in T_{filt}} |t - t_{target}(|T_{list}|)|$
17: until $(|t_{target}(|T_{list}|) - t_{target}(|T_{list}| - 1)| > \Pi) \wedge (|T_{list}| < I)$
18: return $t_{tareget}(x)$ :
        $F(W, t_{target}(x), k) = \min_{1 < i < |T_{list}|} F(W, t_{target}(i), k)$
```

how the simulated workload differs from the real process (using the simulated observed execution times instead of the original r_{ob} values in the function of Eq.1).

Next, line 11-13 searches through the past error values for each timestamp in T_{red}. With the help of the $\phi(x)$ function, we look for t_{min} where the past error function and its simulated counterpart are aligned. Next, the we analyse the future error function: Line 15 minimises the future error around t_{min} within radius $S/2$. The timestamp with minimal future error is t_{target} for the iteration.

Finally, the we repeat until the successive change in t_{target} is smaller than the precision Π or the iteration count reaches its maximum – I. The algorithm returns with the $t_{target}(x)$ value of the iteration x which resulted in the smallest future error $F(W, t_{target}(x), k)$. This returned value then could be reused by the workflow enactor for the planning and execution phases of the autonomous loop, where precise details are out of the scope of this paper.

5 VALIDATION

5.1 Simulating the Tinker workflow

We demonstrate our approach via a biochemical workflow that uses the TINKER library [6] in a Conformer Generator (TCG) workflow which consists of 6 steps (see Figure 1): (*i*) **G**: generating 50000 input molecule conformers (taking around 12 hours, compressed into 20 files); (*ii*) **T1**: minimising the initial conformational states generated at high temperature (35 minutes for a group of 2500 conformers); (*iii*) **T2**: performing a short low temperature dynamics (60 minutes for the group); (*iv*) **T3**: cooling the high temperature states (32 minutes for the group); (*v*) **TC**: collecting parameter study results; (*vi*) **C**: re-compressing results to a single file.

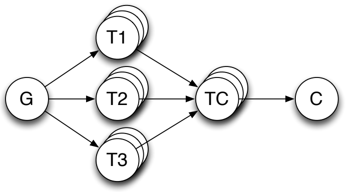


Figure 1: TINKER Conformer Generator workflow

The execution of this workflow was described for the simulation (using the open source DISSECT-CF simulator [7]), which included the model of the cloud of the Laboratory of Parallel and Distributed Systems: an OpenNebula cloud consisting of 216 cores, 604 GBs of memory and 70 TBs of storage. This cloud was used to simulate TCG while various background load was added (we used the Grid Workloads Archive as realistic loads – GWA [8]). Each TCG job had its own VM with 1 CPU core and 1 GB of RAM.

5.2 Evaluation

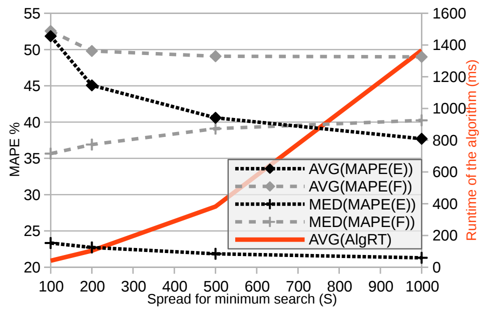
5.2.1 Analysing our assumptions. First, to understand the relation between past and future errors (and thus our main assumption), we simulated the TCG workflow with every starting timestamp from GWA. With videos about Sharcnet[1] and AuverGrid[2], we exemplify how simulated past and future error values vary. Here, each dot represents a single simulation run. We focused on the lower part of the error range (below 10^7) omitting thousands of values (as some reach over $3 \cdot 10^7$) to ensure the best view on the near optimal error values (which are below $2 \cdot 10^6$). Based on our simulated past and future error values, we observed that the prediction could improve if the error values are around the minimum. The videos also show how the error functions converge towards the optimal values as a result of the increasing number of completed jobs – k.

Next, we evaluated the simulated time series of past and future errors. We saw error values between $(1.5 \cdot 10^6 - 4 \cdot 10^7)$. Thus, to find better matching background loads, we limited our search to fragments with past/future error values below a chosen τ threshold:

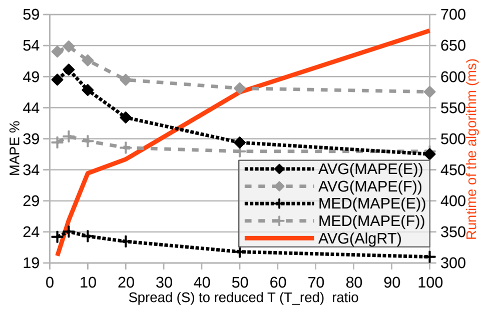
$$T^{exp}_{filt} := \{t \in T : E(W, t, k) < \tau \vee F(W, t, k) < \tau\} \quad (3)$$

To further understand our assumption on the relations of past and future errors, we evaluated how likely consecutive (in terms of t) fragments with small past error values $E(W, t, k)$ lead to small $F(W, t, k)$ values. First, we prepared T^{exp}_{filt} using $\tau := 2 \cdot 10^6$. We selected those subsets that hold more than 80 consecutive timestamps of the trace. Next, we observed that in these subsets the likelihood of having both minimal future and past error values is 65-86%. We also observed that lower τ values notably decrease the

[1]https://youtu.be/gozmHoCneyU
[2]https://youtu.be/BzdVcAq4ez8



(a) S



(b) $(\max T_{red} - \min T_{red})/S$

Figure 2: The influence of the inputs of the algorithm

simultaneous presence of below-threshold error values (suggesting over fitting). Our assumption was proven: past error values indicate the tendency of future ones.

5.2.2 *Selection performance and convergence.* To validate the capabilities of the algorithm in terms of workload approximation, we first defined the metrics to quantify the accuracy. We randomly selected a fragment from GWA (denoted it as t_g) and used it as the workload to be predicted behind TCG, then we applied our algorithm to approximate this fragment. We calculate the accuracy of the load approximation of the received t_{target}, by evaluating the Mean Absolute Percentage Error (MAPE) of all possible (in terms of k) past and future execution time error values as follows:

$$MAPE_E(W, t_g) := \sum_{1 \leq i \leq N} \frac{|E(W, t_g, i) - E(W, t_{target}, i)|}{\frac{N}{100} E(W, t_g, i)} \tag{4}$$

for past errors, and for future errors:

$$MAPE_E(W, t_g) := \sum_{2 \leq i \leq N-1} \frac{|F(W, t_g, i) - F(W, t_{target}, i)|}{\frac{N-2}{100} F(W, t_g, i)}. \tag{5}$$

As our algorithm could be influenced its input parameters, we carried out a parameter study. We investigated how does the accuracy and the approximation time d changes using the most influential parameters : (i) the search range – S –, and (ii) the relative size of search window – $(maxT_{red} - minT_{red})/S$. For each particular parameter setup, we have executed 500 random approximations of with the GWA trace. Finally, to quantify the algorithm's quality, we used the average and median values of our accuracy metrics of these 500 approximations. As a baseline, we evaluated the accuracy metrics for random trace selection with the following results: $AVG(MAPE_E(W, t_g)) = 157.874$, $AVG(MAPE_F(W, t_g)) = 166.166$, $MED(MAPE_E(W, t_g)) = 49.18$, $MED(MAPE_F(W, t_g)) = 67.825$.

The algorithm's performance is presented in Figure 2. In total, each figure represents over 500 thousand approximations. It can be seen that in all cases the results of our algorithm yields better results than random selection. Regarding the effects of the various parameters, increasing any of them obviously introduces more calculations and hence, the d duration increases monotonously. Accuracy improves with increasing search range S but it simultaneously increases the execution time (Figure 2a). It is important to notice that the median of $MAPE_F$ increases indicating that despite the improving average values, an increasing number of results lie out of the desired range. Finally, the relative size of the search window generally improves the accuracy with accordingly increasing execution times (Figure 2b).

In general, the duration of the approximation is negligible (in the range of 31ms-2114ms, with the median of less than 200ms) compared to our assumed 1 minute maximum time to be spent on workload prediction. This leaves enough time for the simulation needs of the algorithm.

6 CONCLUSIONS

Scientific workflows are long-running applications thus, modifications in the infrastructure are likely to be necessary at runtime in order to accommodate changes in the load and maintain performance. A key for such runtime adaptation, information about the load, is absent in most of the cases. In this paper, we proposed a novel background load prediction algorithm for cloud-oriented workflow enactment. The principle was demonstrated and validated using a biochemical workflow. Tests proved the ability of our approach to select workload traces that are suitable for predicting background load. In our future work, we consider broadening the scope of our predictions from private clouds (that could be easily modelled in feasible time with current simulators) to some commercial clouds as well.

REFERENCES

[1] P. Horn. Autonomic Computing: IBM's Perspective on the State of Information Technology. *IBM TJ Watson Labs.*, October 2001.

[2] Rodrigo N Calheiros, Enayat Masoumi, Rajiv Ranjan, and Rajkumar Buyya. Workload prediction using ARIMA model and its impact on cloud applications' QoS. *IEEE Transactions on Cloud Computing*, 3(4):449–458, 2015.

[3] Deborah Magalhaes, Rodrigo N. Calheiros, Rajkumar Buyya, and Danielo G. Gomes. Workload modeling for resource usage analysis and simulation in cloud computing. *Comput. Electr. Eng.*, 47(C):69–81, October 2015.

[4] E. Caron, F. Desprez, and A. Muresan. Forecasting for grid and cloud computing on-demand resources based on pattern matching. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 456–463, Nov 2010.

[5] Gábor Bacsó, Ádám Visegrádi, Attila Kertesz, and Zsolt Németh. On efficiency of multi-job grid allocation based on statistical trace data. *Journal of Grid Computing*, 12(1):169–186, 2013.

[6] A. Kertesz, F. Otvos, and P. Kacsuk. A case study for biochemical application porting in european grids and clouds. *Special Issue on Distributed, Parallel, and GPU-accelerated Approaches to Computational Biology, Concurrency and Computation: Practice and Experience*, August 2013.

[7] Gabor Kecskemeti. DISSECT-CF: a simulator to foster energy-aware scheduling in infrastructure clouds. *Simulation Modelling Practice and Theory*, 58P2:188–218, November 2015.

[8] Alexandru Iosup, Hui Li, Mathieu Jan, Shanny Anoep, Catalin Dumitrescu, Lex Wolters, and D. H. J. Epema. The grid workloads archive. *Future Generation Comp. Syst.*, 24(7):672–686, 2008.