Credit-based algorithm for Virtual Machines Scheduling

Andreas Pegkas
Dept. Computer Engineering and Informatics
University of Patras
Patras, Greece
pegkas@ceid.upatras.gr

Christos Alexakos
Dept. Computer Engineering and Informatics
University of Patras
Patras, Greece
alexakos@ceid.upatras.gr

Spiros Likothanassis
Dept. Computer Engineering and Informatics
University of Patras
Patras, Greece
likothani@ceid.upatras.gr

Abstract—Cloud computing involves a set of new technologies that permit the on-demand utilization of computational resources. This is extremely useful, especially when performing high demanding analysis tasks. The flexibility and scalability on providing computational resources has set cloud computing as an emerging technology for data analysis, where problems demand parallelization and are of high computational workload. In this article, a novel Credit-based algorithm targets at providing an efficient and simple (easy to implement) solution to computational resource management challenge by addressing the virtual machine scheduling problem of serving continuously submitted and different analysis tasks in private cloud infrastructure.

Keywords—cloud computing, resource management, job scheduling

I. INTRODUCTION

Cloud computing involves a set of new technologies that permit the on-demand utilization of computational resources (storage, cpu, memory). This is extremely useful, especially when performing high demanding analysis tasks. The flexibility and scalability on providing computational resources has set cloud computing as an emerging technology for scientific research, where problems demand parallelization and are of high computational workload [1].

Cloud computing is a relatively new and prominent set of technologies in the area of ICT and it is anticipated to play a key role in the modern information systems. In cloud computing, all available computational resources (such as networks, storage, applications, servers, etc.) are provided via web as utilities. The main difference of cloud computing against traditional approaches can be summarized to the on-demand access to the resources pool and the flexible and adaptable resources provision management by the cloud provider. In cloud computing, users can increase or decrease the capacity and consume as many resources as they need for their applications. Furthermore, the technologies composing cloud computing can be combined in order to provide highly scalable and adaptable applications, owing to the fact that new applications and features can be deployed significantly faster. Another worth mentioning characteristic of cloud computing is that all the provided resources are hosted on remote server infrastructures, allowing them to be accessed remotely as long as there is Internet connection and users and systems can collaborate simultaneously on these resources [2].

Nowadays, most research institutions have established private clouds, i.e. cloud infrastructure that resides inside their premises and is operated only by their staff and researchers. Although private clouds are considered more secure against public clouds [3], in most cases they have limited resources, thus making the continuous execution of high complexity analysis tasks a time-critical task.

In the current article, we are focusing on the problem of efficient management of the available computational resources in a limited cloud computing infrastructure. More specifically, the article focuses on the real-time scheduling of the execution of jobs that require specific computational resources (memory, CPU, storage) on a private cloud. The proposed Credit-based algorithm targets at providing an efficient and simple (easy to implement) solution to the job scheduling problem in the cloud computing. The application of this solution could be in a system that accepts submission for analysis tasks and using the proposed algorithm can easily manage the creation/deletion of the Virtual Machines (VMs) that will finally execute the analysis and ensure that the minimum required resources in terms of performance and cost will be used.

The proposed approach is presented in detail in the rest of the paper. In Section II algorithms that are already used for job scheduling in the cloud are presented. Section III describes the proposed Credit-based Algorithm. Section IV presents the simulation experiments that are conducted for the evaluation of the proposed algorithm, while the results are presented in section V. A discussion around the results is depicted in section VI. Finally, Section VII concludes the article and presents future work.

II. RELATED WORK

A lot of algorithms have been proposed for job scheduling in a cloud computing infrastructure [4][5][6]. In this section we present the most prominent of them.

For each of the algorithms, a running example is presented for a small indicated scenario. The jobs are submitted in the systems with the following order: JOB1(1,18), JOB2(2,5), JOB3(6,5), JOB4(4,5), JOB5(2,15), JOB6(1,10). In job notation the first number in the brackets denotes the number of
resources (in nodes) required for job execution and the second number the time duration of the execution.

A. First-Come-First-Serve

First-Come-First-Serve (FCFS) is a classic scheduling algorithm. According to FCFS, all the jobs are submitted to a queue. The first job in the queue is served when there are the available resources. If there are not available resources the system is waiting until a running job finishes and its resources become free. Unfortunately, this approach usually underutilizes the system while the majority of the resources may be unexploited for a high amount of time. Also there is the case of jobs that require a lot of resources that usually wait for a long time in queue delaying the execution of the total of jobs. Fig. 1 presents the jobs’ execution for the indicated scenario for FCFS.

B. Conservative and Aggressive Backfilling

Backfilling [7] requires the a-priori knowledge of the execution time of the job along with the required resources needed. The logic behind Backfilling is that “depicts” a job as a rectangular in a two-dimension world where the first dimension is the execution time and the second is the amount of required resources. If we put in a timeline the execution of the jobs in the cloud computing infrastructure, each job will occupy a specific space part. The blank spaces between the jobs can be used for the execution of jobs with low requirements in time and resources, even if submitted later in the queue.

There are two alternative Backfilling algorithms:

• Conservative Backfilling

• Aggressive Backfilling (EASY)

In the Conservative Backfilling the jobs are scheduled according to their submission order if there are available resources. Otherwise, the algorithm searches for “small” jobs in the queue in order to execute them earlier. Each job has a reserved area of time/resources (backfilling). Also, if this reserved area contains other jobs, which have been placed there by accident because they were characterized as “small”, the later are stopped.

Aggressive Backfilling is similar to Conservative Backfilling, with the difference that only the first job in the queue has the privilege to reserve an area of time/resources and to stop other jobs that exist in this area. This creates a disadvantage against Aggressive Backfilling, while there is no guarantee for the response time of all the jobs.

Both Conservative and Aggressive Backfilling algorithms utilize users’ estimation regarding job execution time, as it is impossible to calculate the execution time of a job before it starts running. Nevertheless, these estimations, in most cases, tend to have a large deviation from the actual, so the resulting calculations are wrong. Fig 2 depicts the jobs execution in the indicated scenario for EASY. EASY was selected against Conservative Backfilling as it is less complex and thus it is widely used in implementations.

C. Conservative Migration supported Backfilling

The Conservation Migration supported Backfilling (CMBF) algorithm [8] is based on the backfilling technique. It is assumed that all jobs can be paused and resumed at a later time and in a different processor and memory.

The CMBF initiates jobs based on their order of submission, if there is sufficient amount of computing resources. When there are not enough resources available to perform a job, a subsequent work with lower requirements on backfilling computational power may begin. To avoid the case that a job that is rather demanding in computational resources is never performed, CMBF follows the following tactics: A previous job, which is not yet running, has the ability to start when the sum of the number of computing resources that are free and those used in subsequent job are equal to or greater than required. In this case, previous job has the potential to "stop" any subsequent job needed to start itself. These jobs store their state, stop running, and return to the queue for future execution. The scenario of CMBF execution is presented in Fig. 3

D. Aggressive Migration supported Backfilling

In the worst case scenario, CMBF is required to store backfilling information for each queued job. This cost can be very large when the queue grows in size. For this reason,
Aggressive Migration supported Backfilling (AMBF) [8] was designed which is more simplified and addresses this problem.

Unlike CMBF, AMBF maintains a backfilling list only for the first queue job and only gives this ability to stop other jobs to run itself. The rest of the queue can only start running if there are enough available computational resources. In addition, another advantage of the AMBF is that it causes less work pauses than the CMBF, achieving a smaller percentage of migrations. The results of executing the scheduling scenario with AMBF are presented in Fig. 4.

![Fig. 4. Job scheduling example of AMBF](image)

### III. CREDIT-BASED ALGORITHM

The proposed Credit-based algorithm, which is based on previous algorithms presented by authors in [9][10], performs workload management by creating a system in which each job enters the queue marked with a value, depending on its computational requirements. As computational requirements we may refer to the number of processors, the total amount of RAM, the number of virtual machines of predefined sizes or some combination of the above that is required for the job execution.

This scheduling algorithm behaves as fair as possible, especially against a simple FCFS scheduling algorithm. With the FCFS algorithm even if there are available resources for a newly coming job, it has to wait until previously submitted larger jobs are executed. On the other hand, if a priority is given to “small” jobs there is the danger a “bigger” job waiting for a long time or even forever. In order to minimize this injustice, we conducted a credit-based fair queuing algorithm. Credits are accumulated to each job when they initially enter the queue. The credits of each job are different and based on their computational and memory requirements. The higher the requirements, the higher the credits are. Credits are spent one every timeslot while a job is waiting in the queue. The jobs with the lower amount of credits have the highest priority. Among jobs with the same credits, the earliest submitted one is selected for execution.

As example, let us assume that we have two different jobs in the queue Job1 and Job2. Job1 has fewer requirements than Job2. Accordingly, Job2 has more credits than Job1. This indicates that Job1 will be executed first. While Job1 is executed, Job2’s credits are reduced. Before Job1 finishes, a new job (Job3) comes in the queue. It has similar requirements with Job1. In case that it has more credits than Job2, Job2 will be the next to be executed. In case that Job3 has fewer credits than Job2, Job3 will be the next to be executed. In case that both Job2 and Job3 have the same credits, Job2 will be executed first. In this manner, jobs with small requirements usually have a priority against jobs with big requirements, but not leading big jobs to starvation is taken into account.

![Fig. 5. Flowchart diagram of the proposed algorithm](image)

In detail, according to Credit-based algorithm, when a job is submitted to the queue, the algorithm assigns a value (credit) to the job. This value is calculated based on the computational requirements of the corresponding job. The larger these are, the greater is the initial credit assigned to it. Fig 5 shows the flow of the algorithm and its steps numbered and explained in the next paragraph.

The algorithm is executed periodically (e.g., every 1 minute). In each execution, the jobs are sorted in ascending order based on the credit value of each (from least to highest) (1). Then a check is performed whether the computational resources needed by the first job (2) are available. If so, job execution starts, it is removed from the queue (3), and the same procedure occurs for the next job (4).

In case a job that is being checked cannot be executed yet, because of unavailable resources, checking is stopped for all the remaining jobs in the queue (2). In the end, the credit values of jobs still in the queue are reduced (5). This is done based on a predetermined rate of reduction (e.g., 1, 5, 0.1, etc.). The minimum credit value that a job can have is zero and in case of same price, the one that entered the queue earlier is preceded.

Credit-based algorithm is intended to be simple in order to be easily implemented and adapted in existing recourse management systems in cloud computing infrastructures. The simplicity of the algorithm can be concluded from its pseudocode that is depicted next.
Credit Based Algorithm

\[ \text{Input: } Q: \text{ the queue for incoming jobs;} \]
\[ M: \text{ a map between jobs and nodes;} \]

\[ \begin{align*}
\text{begin:} \\
& \quad \text{sort the jobs in } Q; \\
& \quad j \leftarrow \text{get the first job from } Q; \\
& \quad \text{while } j \neq \text{NULL do:} \\
& \quad \quad N_j \leftarrow \text{the number of nodes required by } j; \\
& \quad \quad N_{\text{idle}} \leftarrow \text{the number of idle nodes;} \\
& \quad \quad \text{if } N_j \leq N_{\text{idle}} \text{ then:} \\
& \quad \quad \quad \text{remove } j \text{ from } Q \text{ and dispatch;} \\
& \quad \quad \quad \text{it to any } N_j \text{ idle nodes;} \\
& \quad \quad \quad \text{update } M \text{ accordingly;} \\
& \quad \quad \quad j \leftarrow \text{get the next job from } Q; \\
& \quad \quad \text{else} \\
& \quad \quad \quad \text{break;} \\
& \quad \quad \text{fi;} \\
& \quad \text{done;} \\
\end{align*} \]

As already mentioned, the proposed credit based algorithm is intended to initially give priority to tasks with lower resource requirements, without trapping the larger ones inside the queue for too long time (starvation).

The reasons why priority is initially given to jobs with small resource requirements are the following:

- By prioritizing serving of smaller tasks, the number of jobs that can be served simultaneously is increased (lower queue). This way we achieve a greater number of users who are satisfied.

- It has been observed that smaller jobs can be very long in time, but in general they tend to be shorter than those that require more computational resources [11].

In the way the graduation of jobs is performed, priority is generally given to small jobs, but when a very large job has been waiting for a long time and its credits have been reduced to zero, it will be the first priority in execution, since all preceding jobs have already started their execution. This may create a larger queue but is necessary if we want to avoid indefinite wait (starvation) for it.

IV. EXPERIMENTS

For the evaluation of the proposed algorithm a number of experiments in simulation mode has been executed. The performance of the proposed Credit-based algorithm tested against the FCFS, EASY, AMBF, CMBF algorithms. The preparation of the experiments included the task of creation of simulation workloads. Afterwards, the workloads have been created, each algorithm, which was developed in python, has been executed against these workloads.

A. Workloads

In order to better compare the above-mentioned algorithms with the proposed algorithm, mathematical models were used to generate workload data sets based on a study of real-world work in computer centers. The aim of these models is to produce artificial sets that are actually based and can be used in parallel processing simulations. Although, there are no such models for cloud computing, older mathematical models for mainly distributed resources were used. In the next paragraphs we present two models which have been proposed after extensive study of real sets, and which have been used in the experimental procedure.

1) Feitelson Model

The Feitelson96 model [11] is based on the observation of the following different parallel processing machines, such as 128-node iPSC/860, NASA Ames, 128-node IBM SP1, Argonne National Laboratory, 400-node Paragon, San Diego Supercomputer Center (SDSC), 126-node Butterfly, Lawrence Livermore National Laboratory (LLNL), 521-node IBM SP2, Concurrent Technologies Corporation (CTC), 96-node Paragon, ETH, Zurich.

The aim of this model is to incorporate a variety of features from real sets that may not be mathematical models. In particular, the Feitelson model takes into account the distinct division of tasks, the correlation between the degree of parallelism and the run time and the repeated execution of some tasks.

Through the study of the log files of the aforementioned parallel processing machines, two phenomena were observed and they were modeled. Small jobs are much larger in number than large ones, and some work sizes are preferred over others with more popular sizes than 2.

Variability of execution times is high, ie the range of resulting times is very large and there is no particular concentration around a certain value. In addition, there is a weak correlation between work size and run time. Although there are small jobs with long duration and large jobs with short runtime, larger jobs tend to be running for longer.

Although there is the assumption that all the works are independent of each other, the analysis of the logs shows that this is not always true. In fact, there are many instances where users perform the same work multiple times, thus creating a sequence of tasks with similar properties. Incorporating this feature into this model results in the data set being given feedback from the program that manages the execution of tasks, as a job that is similar to the previous one begins to run only after it is terminated. As a result, if a repeated work is delayed, all tasks similar to this are delayed.

2) Jann Model

The Jann97 model [12] is a statistical model based on an actual set of tasks belonging to the Concurrent Technologies Corporation (CTC) 322-node SP2 parallelepiped from June 25, 1996 to September 12 1996. During that period, 17440 jobs were carried out.
This model is based on finding hyper-erlang distributions for types of jobs that were executed according to their size and run time. The hyper-erlang distribution is a generalization of the erlang distribution.

3) Tsafrir Model

The model Tsafrir05 [13] aims to generate user estimations for a task’s execution time. These runtime estimations by users are necessary in simulating backfilling based algorithms. These values are also necessary when studying backfill algorithms because they have a lot of deviation from the actual ones. This results in the performance of algorithms getting worse. But if we want to study them, we need to have a realistic picture of the results that cannot be given if we use the exact execution times, which in actual circumstances cannot be known in advance.

The time estimation values produced are not chosen in a random fashion but are created based on some empirical rules that have been observed in real-world conditions. The main rules on which the prices are calculated are the following

- The distribution of prices is very distinct, ie users tend to choose specific values such as 5 minutes, 1 quarter, 1 hour. In fact, about 90% of the total estimates consist of a small set of different values.
- In the majority of cases the most popular price is the maximum possible provided by the system. When a task exceeds this time, it automatically shuts down the system.

B. Experiments

Initially, the workloads that would be used as input were selected. Since it was not possible to find real sets of work that respond to conditions prevailing in large computing centers, it was decided to use model generated data-sets. The models of Feitelson96 and Jann97 were used to create two different sets of tasks for which they were executed, each separately, and routed using all the implemented routing algorithms. The Tsafrir05 model was applied to the produced workgroups of the other two to add user predictions. These predictions were necessary for executing the backfilling algorithm EASY.

The models provided by the authors were used to implement each workload with some minor modifications. These models implemented in C++ programming language, and it is necessary to use either special tools (compilers) on a Windows operating system or to use a Linux operating system.

The complete code of the algorithms as well as the simulation of the execution of the work was carried out in the context of the preparation of the postgraduate thesis in Python programming language. This language was judged to be the most ideal based on the nature of the code that had to be implemented.

V. RESULTS

During the experiments execution process, two sets of workloads were simulated. The first workload set is based on the Feitelson96 model and other is based on the Jann97 model. For each model, a set of 10 different workloads with different parameters was created and used for the experiments. Each of these sets was time scheduled with the use of five different algorithms. In addition, for the proposed algorithm, the workloads were scheduled with four different configurations in order to find the optimum one.

The metrics used to compare the performance of the algorithms are the following:

- **Response time**: the time elapsed from the time the job is added to the queue until the time it starts to be executed.
- **Utility**: the percentage of available computer resources that is used on average at any time.
- **Finish time**: the time from the start of executing jobs to the completion of the last job.

The results are presented in the following paragraphs. As final values for the metrics the average of each set of workload is used.

A. Algorithm Comparison for the Feitelson Model

After the execution of time scheduling with each algorithm for the Feitelson-based workload set, the following results were obtained for each algorithm about the finish time, the utilization rate of available resources and the average response time:

![Fig. 6. Finish time of all algorithms (Feitelson workload)](image)

According to the two graphs above, it is noticeable that the metric values of finish time and utility have very small difference, so no safe conclusion can be drawn.

![Fig. 7. Utility of all algorithms (Feitelson workload)](image)
In the above graph it is observed that the proposed algorithm achieves a fairly good performance, which is much better than that of the simple but widespread FCFS and is close enough to the performance of the more complex EASY, AMBF and CMBF. The graph that is presented below shows the results of the individual averages per job size, as well as a table of results.

The performance of the proposed algorithm is a little worse compared to the other complex algorithms but has a much simpler implementation than the rest. The EASY algorithm, in addition to the high complexity of algorithmic calculations, requires prior knowledge of the execution time of each job, which is not feasible, and instead uses the predictions of users that are usually highly inaccurate.

AMBF and CMBF do not require any extra information, but they have as a prerequisite the ability to pause and restart jobs during execution from the same state. This makes the implementation extremely difficult because it requires not only to classify the jobs in the system, but also to interfere with the operation of virtual machines. In addition, in some cases virtual machine pausing and restarting later on is virtually impossible, as this requires full storage and retrieval of RAM data which has been used in the meanwhile to perform other tasks.

B. Algorithm Comparison for the Jann Model

After the execution of time scheduling with each algorithm, for the Jann-based workload set, the following results were obtained for each algorithm about the finish time, the utilization rate of available resources and the average response time:

As earlier, it is observed that in the above two graphs the metric values of finish time and utility have very small difference, so no safe conclusion can be drawn.

What can be noticed in the above graph is that the proposed algorithm achieves a fairly good performance, which is better...
than the simple but widespread FCFS, but worse than the performance of the more complex EASY, AMBF, and CMBF. This happens because in this workload model the percentage of very small in resources jobs is even greater and close to 90%. However, in order to have a better picture of the average times of execution of the jobs, a graph is presented in the next figure with the results of the individual averages per job size, as well as a table with the results.

Fig. 13. Response time of all algorithms categorized by the resource related size of the jobs (Jann workload)

What is remarkable is that the EASY algorithm can achieve much better performance at the average of all jobs, but for the very resource-demanding jobs, the response time is prohibitive as it spins to huge numbers and is very likely the corresponding job to be canceled by the user before they start its execution. In addition, a high complexity of algorithmic calculations is required, and prerequisite is the prior knowledge of run times, which is not feasible, and instead, the predictions of users are used which are not accurate.

Finally, the AMBF and CMBF algorithms generally have better performance at all levels and do not require some extra information, but they have high complexity and additional conditions that may hinder their implementation under real circumstances.

VI. DISCUSSION

What can be deduced from the analysis of the experimental results is that the proposed algorithm, which is based on credit rating according to the amount of computing resources that are required, but also based on the time each job has been in the queue, achieves satisfactory results.

Performance of the proposed algorithm is much better than that of FCFS, which does not achieve good results but is a widespread algorithm due to the simplicity of its implementation. The advantage of the proposed algorithm is that it performs much better but has a relatively low implementation and execution complexity, like the FCFS.

EASY backfilling algorithm in Feitelson workload model has slightly better results than the proposed credit based, while in Jann workload model it achieves the best average time for all the jobs, but in very high-demand jobs the time delay is prohibitive. Although this algorithm has been present for some time, it is not often used in real environment, as it has very high computational complexity, and in order to function optimally there should be prior knowledge of the execution time of each task, which is usually not possible. Therefore, in real cloud computer systems the proposed algorithm is best suited for use.

Finally, AMBF and CMBF algorithms perform better at all levels and do not require some extra information, but they have as prerequisite the system to have the ability to pause and restart virtual machines from the same state. Consequently, their implementation becomes difficult because they do not simply categorize jobs inside the system, but they modify virtual machine operation. In addition, in some cases pausing virtual machines and restarting them later on is virtually impossible, as this requires full storage and retrieval of RAM data, which has been allocated over time to perform other tasks. So they have theoretically much better results but cannot be applied to real cloud computing centers. Additionally, in all metrics performed, the extra time required to pause and restart virtual machines has not been calculated, which is not negligible but is calculated in a few minutes.

VII. CONCLUSION AND FUTURE WORK

In this article, a novel Credit-based algorithm targets at providing an efficient and simple (easy to implement) solution to computational resource management challenge by addressing the virtual machine scheduling problem of serving continuously submitted and different analysis tasks in private cloud infrastructure. Although the proposed algorithm does not achieve the best results in all cases, it is much easier to apply in real conditions, and performs better than the ones that already applied, making it more attractive for utilization in real world systems.

In a future research, the proposed scheduling algorithm could be applied to a real cloud computing. The behavior of the algorithm and its performance could be further studied. This way, it is possible to further optimize the parameters used to execute it.

In addition, variants of the algorithm can be created where the queue priority shift changes based on some additional rules such as the size of the job queue or even the type of jobs that are on hold.

Still, if it is proven that in the future virtual machines pause and restart is easy to implement and has a low migration cost, a corresponding mechanism could easily be applied to this algorithm. Opportunities for optimization are many, provided there is mood and ingenuity.

REFERENCES


