2014 40th Euromicro Conference on Software Engineering and Advanced Applications

Profiling-Based Task Scheduling for Factory-Worker Applications in Infrastructure-as-a-Service Clouds

Rostyslav Zabolotnyi*, Philipp Leitner†, Schahram Dustdar*
*Distributed Systems Group, Vienna University of Technology
Argentinierstrasse 8/184-1, 1040 Vienna, Austria
{rstzab, sd}@dsg.tuwien.ac.at
†s.e.a.l. - software evolution & architecture lab, University of Zurich,
Binzmühlestrasse 14, 8050 Zurich, Switzerland
leitner@ifi.uzh.ch

***Abstract*—With the recent advances of cloud computing, effective resource usage (e.g., CPU, memory or network) becomes an important question as application developers have to continuously pay for rented resources, even if they are not used effectively. In order to maintain required performance levels, it is currently common to reserve resources for peak resource usage or possible resource usage overlaps, if more than one task is executed on a host. While this is a reasonable approach for long-running applications or web servers, for some applications with disperse resource usage over time, this strategy causes significant over-provisioning and thus resource wastage and financial loss. In this paper we present a profiling-based task scheduling approach for factory-worker applications that schedules tasks within the defined resource limitations (e.g., existing machine memory size or network quota) and distributes the tasks in the cloud environment in order to use resources effectively. The evaluation of our approach approved the efficiency of the proposed algorithm and minimal performance overhead. In case of the evaluated application, the presented scheduling process leads up to 33% resource savings with only 1% of performance loss.**

Keywords*—*Profiling, Cloud computing, Scheduling, Factory-worker, Infrastructure-as-a-Service, Resource usage, Elasticity

I. Introduction

Cloud computing [1] brings new opportunities to developers and allows to simplify application scaling and resource provisioning. The benefits of cloud computing are widely recognized nowadays, but cloud computing is not a silver bullet and does not fit every use case. Even more, depending on the way it is used, the benefits and costs may differ dramatically. Cloud computing is a promising technological choice for companies that are struggling to keep up with resource-heavy, computation-intensive or high request rate application demands, but depending on the way the application is using the cloud, costs and performance may vary significantly.

In addition to the benefits of virtually unlimited on-demand storage and processing capacity, cloud users have to be aware of additional costs that might not be visible in the local data center or during local prototype development. For example, cloud resources are assumed to be always available and accessible on demand, but in reality starting up a virtual machine may take several minutes or fail entirely [2], the performance of two identical virtual machines may vary [3], and virtual machines can crash at any time. Cloud applications need to be equipped to handle such situations. One common solution to some of these issues is to predict application load and reserve resources before they are actually needed. However, it is not clear what the application should do in situations of rapid demand change or variable resource requirements. For example, to handle a brief resource-intensive task, it may be infeasible to start additional cloud hosts, schedule workload there, and shut them down immediately when a task is processed.

In this paper we tackle the question of efficient and elastic cloud resource usage. Usually, tasks executed within the cloud do not use resources (e.g., memory, CPU or network bandwidth) uniformly. Instead, over the task run, resource usage varies, causing usage peaks and valleys. In order to achieve effective and predictable execution times, developers have to reserve resources considering the maximum expected usage [4]. This causes resource over-provisioning for, often significant, parts of the task execution time. If multiple tasks are to be processed on the same machine, resource over-provisioning is even higher, as developers have to reserve resources accordingly to the worst-case scenario, when resource usage peaks overlap. For instance, to process multiple tasks in the cloud, with a 1GB peak memory usage each, developers have to either use hosts with 1GB of RAM, and execute tasks sequentially on each host, or reserve machines with more memory, thus allowing parallel task execution. However, if this memory usage peak takes only a short period of the task processing time (e.g., during data serialization), while remaining memory usage is much lower, all reserved memory for that peak demand is wasted most of the time. Here, we present a cloud resource scheduling approach that allows effective resource usage for uniform tasks based on profiling data. We developed a scheduler that monitors task execution and constantly improves future resource usage estimations for each used host. These predictions allow the effective scheduling of subsequent tasks and forecasting when application should scale up or down, thus improving elastic system behavior in the cloud and optimizing resource usage. Additionally, the presented Scheduler aims to avoid overlapping peak resource usages of tasks, hence allowing to run more tasks in parallel on the same virtual machine.

Different types of applications require different approaches to scaling and monitoring. The scheduling approach proposed in this paper is most useful for factory-worker applications (also known as the producer-consumer pattern, and strongly

978-1-4799-5795-8/14 $31.00 © 2014 IEEE
DOI 10.1109/SEAA.2014.42

related though not identical to the master-slave pattern [5]). In factory-worker, a single host or a set of hosts (named "factory" hosts) create tasks while a (typically large) number of worker hosts process them. This architectural pattern is commonly used in situations where the system has to process a set of tasks generated from user requests or by splitting the bigger problem into smaller chunks. Applications designed this way often achieve high scalability and performance while keeping interaction code simple. These advantages make the factory-worker architectural pattern a common choice for applications that run in a distributed environment or the cloud. Another distinctive feature of factory-worker systems is that the set of possible tasks is usually homogeneous or limited. This allows predicting future resource usage basing on previously gathered profiling data. This characteristic makes the scheduling approach presented in this paper particularly useful for factory-worker applications.

The remainder of this paper is structured as follows. Section II describes the motivational application that illustrates the need for a resource usage-aware scheduling algorithm. Section III describes research related to our work and how it is different from our approach. Afterwards, in Section IV, we present a resource-aware scheduling algorithm, which is the core contribution of this paper. This algorithm is consequently evaluated on the sample application in Section V. Finally, the paper is concluded in the Section VI.

II. Motivating Scenario

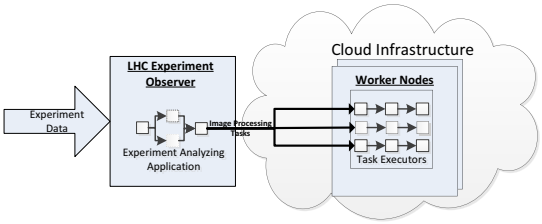


Figure 1: LHC experiment results processing application using cloud infrastructure

The "Large Hadron Colider" (LHC), built by the European Organization for Nuclear Research (CERN), is considered as "one of the great engineering milestones of mankind" [6] and allows physicists to test predictions of particle and high-energy physics. While this project solves important problems in physics, it generates some interesting challenges for computer science, big data and distributed computing as well [7]. One of these challenges is a problem of image processing, that deals with images that are being captured during collision experiments. During each particle collision experiment, 100-megapixel cameras take 40 million pictures a second, producing 1 petabyte (PB) of image data per second [7]. As the experiments that cause these measurements are periodic and require significant time of preparations and equipment testing, this challenge is a suitable use case for cloud computing. Instead of continuously operating a sizable data center [7], CERN can rent the required amount of cloud machines on demand, only when resources for image processing are necessary.

This task is a specific instance of the more general pattern of a factory-worker application, as discussed above (see Figure 1). While we do not have any specific information on the resource usage of the image processing algorithms used by CERN, we can assume that processing of a similar set of images should produce similar resource usage patterns. Additionally, we can assume that the resource usage is also not uniform over the entire time of processing a single image (e.g., there may be significant network bandwidth usage at the beginning and at the end of the execution, and memory spikes for image decoding). Therefore, when the tasks are to be scheduled to the cloud infrastructure, resources have to be reserved considering the worst-case scenario (that all images handled by a host are e.g., in the decoding phase at the same time), causing resource over-provisioning. This is illustrated by Figure 2 for memory usage. In this paper we consider an approach for improving such scenarios, achieving more effective resource usage without significant performance loss.

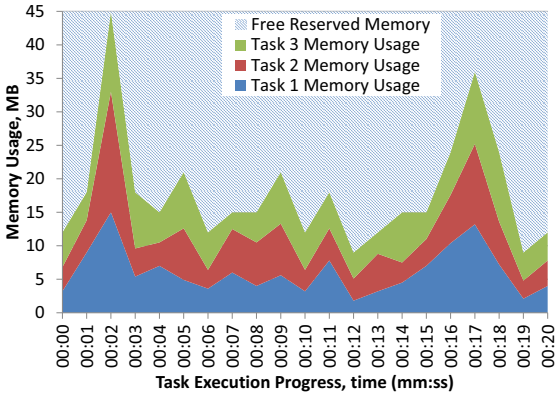


Figure 2: Host Memory usage in case of memory peaks overlapping

III. Related Work

Effective scheduling in cloud computing domain is a vital topic, as it influences dramatically the overall performance of the application as well as resource usage, which is clearly important since the pay-as-you-go model is an intrinsic feature of cloud computing [1]. However, research in the area of cloud scheduling is mainly focused on QoS [8], [9], [10] cost-awareness [11], [12], [13], [14], [15] or SLA-conformance [12], [16], [17]. Each of these approaches operates on higher levels of business requirements and service quality, rather than effective resource usage, as the one presented in this paper. Relatively new, but nevertheless a quite prominent topic in cloud computing is energy-efficiency or "green" scheduling [18], [19], [20], [21], that focuses more on effective power usage and operation in data centers or application adaptation in order to require a minimal amount of resources. Another scheduling approach that focuses more on reliability and guaranteed task completion in faulty environments is redundant scheduling [22], [23], [24].

From the perspective of profiling, in the cloud computing domain it is common to use it in order to develop elastic applications [25], [26], [27] that adapt themselves to varying loads. This is usually treated in a broader way than the approach presented in this paper: usually profiling is used to

Table I: Resource Types Summary

Resource Name	Measuring Units	Resource Type	Description
CPU	operations per second	Competitive	Specifies application execution speed within existing environment.
Memory	bytes	Cumulative	Specifies amount of used memory within machine.
Network Traffic Usage	bytes	Cumulative	Specifies amount of data transferred over the network.
Network Bandwidth	bytes per second	Competitive	Specifies current throughput of the network.
Storage	bytes	Cumulative	Specifies amount of occupied storage within available disk space.
Storage Read/Write	bytes per second	Competitive	Specifies speed of read/write operations of the storage.
Database Read/Write	transactions per second	Competitive	Specifies amount of successful transactions between system and database.

determine under-utilized or over-utilized machines, in order to balance existing load or manage the amount of used resources.

IV. Resource-Aware Task Scheduler

In this section we describe our scheduling approach and explain how it achieves the outlined goals.

A. Resource Types and Control Limitations

At first, before diving into the details of our approach, it is necessary to discuss the nature of different computational resources and possible ways to control their usage. On the highest level of abstraction, cloud-provided computational resources (see Table I) can be divided into two classes: *competitive* and *cumulative*.

On the one hand, *cumulative* resources can be profiled and predicted relatively easily and confidently. For example, if memory allocation is required and we are handling multiple tasks in parallel, we can assume that the total amount of used memory is the sum of each task usage. However, it is not trivial to reduce cumulative resource usage at a specific point of time. For example, when some task will need more memory than available in the system, we cannot reduce memory usage of other tasks, therefore, we have to suspend our task until total memory usage decreases. Additionally, whenever we suspend the execution of a task, the usage of cumulative resources remains constant (i.e., the usage of cumulative resources does not decrease when suspending a task), limiting the effectiveness of task suspension for such resources.

On the other hand, *competitive* resources are easy to manage with task suspension and resuming. For example, when we are approaching some timing-critical CPU-intense computation stage of one task, we can suspend other tasks on the same host and thus ensure that all computational resources are allocated to the critical task. However, competitive resources usage profiling is not as predictable as for cumulative resources. For example, when two tasks are competing over the CPU of a virtual machine, their execution time is hardly predictable because of concurrency issues.

Additionally, we need to keep in mind that resource usage on application level is not entirely predictable in practice. For instance, some requests or request sequences can significantly influence the overall usage and productivity of some resource. For example, database queries of different complexity can take different amounts of time. If concurrent requests work with completely distinctive parts of a database, an execution may significantly slowdown due to the frequent cache misses. Such behavior is hard to predict during profiling, therefore, our prediction system relies on a profiling error correction system, which improves prediction accuracy over time.

B. Approach Overview

The system we present in this paper provides fine-grained scalability and adaptability of the application as decisions are based on the actual application behavior and current activities, instead of general resource usage trends as utilized in related approaches [28], [29], [30], [31]. This is achieved by using a specifically designed distributed profiling solution that allows to collect runtime information from the distributed application in the key points of task execution. The global overview of the task scheduling and execution process is presented in Figure 3.

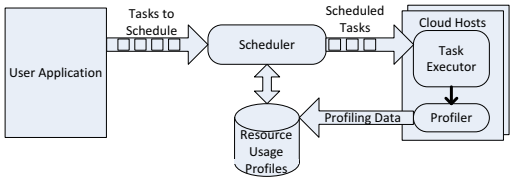


Figure 3: Overview of the profiling-based scheduling approach

To have current and accurate information on application behavior from each used worker (cloud host), the application is monitored via a special Profiling System. The Profiler is running within the application on each used cloud node and collects the information necessary for scaling decisions. Currently, this information includes memory usage by objects related to the tasks executed on this node and CPU usage of the machine. Extension to other types of resources is part of our ongoing work. This information is matched to the tasks executed on this node and their progress, either via push notifications sent by the task that is being executed, or after each task is finished. This system is mainly responsible for information collection, but also handles starting and suspending tasks, in order to prevent overlapping to avoid peak load aggregation. All profile-data processing and the creation of task schedules is happening in a separate Scaling System, which is a conceptually independent component that collects and uses information from Profiling Systems on the workers. The Scaling System is a central planner for our approach, and can be easily deployed to a separate cloud host to not interfere with the worker's performance.

C. Resource Profiling

As described above, the Scaling System relies on resource usage information obtained from profiling previous runs of similar or identical tasks. During each task execution, the Profiling Systems are collecting resource usage traces and periodically send this information to the Scaling System. For a profiled task τ we measure current usage or usage delta (u_i) of each resource ($\forall \rho \in P$). Therefore, the task execution trace

(U) is a mapping of resource usage measurements to the time when the measurement was performed for each measurement point $0..n$, as shown in Equation 1:

$$U \equiv \forall \rho \in P, \forall i \in [0..n] : < t_i, u_i > \quad (1)$$

This trace information, for each resource, can be visually represented as in Figure 4 (the figure exemplifies a trace for memory usage).

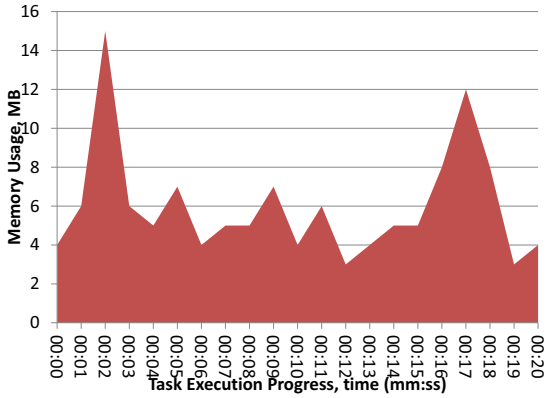


Figure 4: The measured memory usage profile for a specific task execution

After receiving multiple of these task execution traces, the Scaling System can build an estimated average task execution profile ($I_{\tau,\rho} \equiv t \in [t_0;t_x], (\iota_{\tau,\rho,t_0} \ldots \iota_{\tau,\rho,t_x})$) by averaging collected traces (U). After x executions (e) of a task of type τ, we hence end up with x traces for each resource ρ, which we describe via a set $U_{\tau,\rho} = \{U_1, U_2, \ldots, U_x\}$. Each resource usage trace may be not completely accurate and may represent only partial information or may provide distorted data due to external system activity, network problems, or other unpredictable events. To improve the predicted profile, all separate traces must be compared and analyzed to minimize side-effects and minimize statistical errors.

For example, for memory execution profile ($\rho = \text{memory}$), a statistically plausible way to build the average is to take the arithmetic mean of all traces for each point in time t, as shown in Equation 2. As the calculation of each point is generally independent, the algorithm does not have to wait for the whole trace to become available (e.g., wait for all executions to finish). Instead, the average execution profile can be calculated and further improved with each new measurement point of next task executions ($u_i \in U_{\tau,\rho}$), if the measured value diverges from the predicted one more than a configured error rate (ξ). The graphical representation of trace averaging is shown in Figure 5.

$$I_{\tau,\rho} \equiv \forall t \in [t_0;t_x] : \iota_{\tau,\rho,t} = \frac{\sum_{U=U_1}^{U_x} u_t}{x} \quad (2)$$

This estimated profile allows to predict the future load for new tasks of type τ. Therefore, if the scheduling infrastructure knows the current execution point of each task on a specific worker in the system, the Scaling System is able to estimate the future load of each worker and to adapt the task execution schedule to fit the required resource usage limitations within each host.

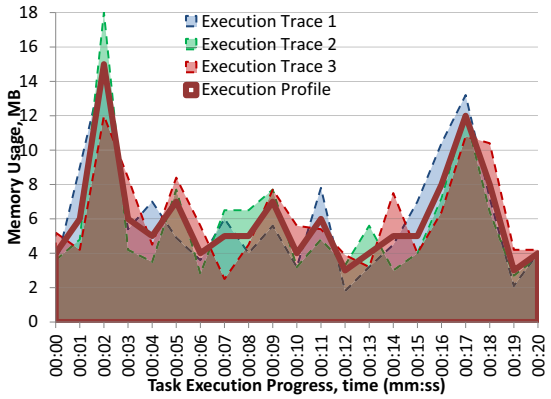


Figure 5: Averaging of measured memory usages to obtain aggregated memory usage profile

D. Resource Profiling Modes

From an implementation point of view, profiling can be performed in *active* or *passive mode*. In *passive mode*, profiling is happening seamlessly to the profiled application. The Profiling System is configured to perform resource measurements in fixed intervals and has no knowledge of the profiled task execution state. This approach gives more freedom to profiled application developers as it does not require any awareness of task profiling. However, it does not provide fine-grained profile information, and may miss some resource usage spikes or misinterpret an application profile because of the interpolation of periodical measurements (see Figure 6). For example, if a task is periodically acquiring and releasing memory, profiling may provide some random memory usage curve because measurements happened on different stages of the periodic process. While an aggregated task profile should improve after multiple task executions, the passive mode is still more appropriate for resources that do not exhibit significant short-term spikes and profiling long-running tasks that only gradually vary their resource usage.

In situations when passive profiling is not appropriate or shows insufficient results, the *active profiling mode* should be used. In this mode, the profiled task is actively triggering the Profiling System to measure resource usage at crucial execution points. This allows obtaining a context-aware resource usage profile that exposes actual task behavior, leading to more confident and reliable scheduling actions. However, this approach has more impact on application performance, requires full awareness of developers and often needs some amount of iterations to achieve required granularity. It is preferable for short-running tasks or applications with short resource usage spikes that can be missed in passive profiling mode. Another important benefit of active profiling is that it allows the Profiling System to suspend task execution on profiling points. This opens up additional scheduling possibilities for the Scaling System, allowing for better resource usage results. In case of passive profiling, task executions, once started, cannot be suspended in our system.

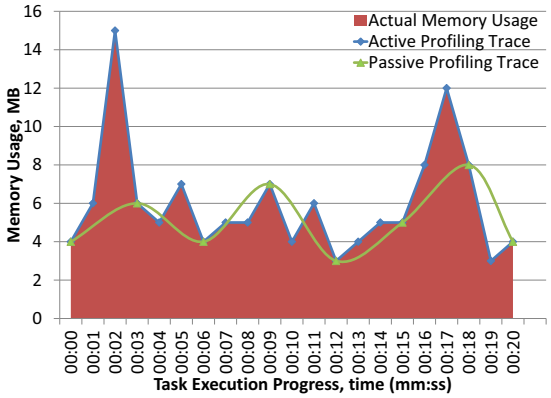


Figure 6: Comparison of Active and Passive Profiling technique on highly dispersing task execution

E. Task Scheduling

Whenever the Profiling System on any worker reports host resource usage (see Figure 7), it also includes information about the execution progress of each currently running task. This allows to scale and align executions of multiple tasks from multiple machines to one *Aggregated Resource Profile* for each distinctive task type existing in the profiled application (see Figure 5). In addition, this profile is further refined to correspond to new measurements, thus improving the overall quality of prediction and adapting the Aggregated Profile if resource usage changes gradually over time.

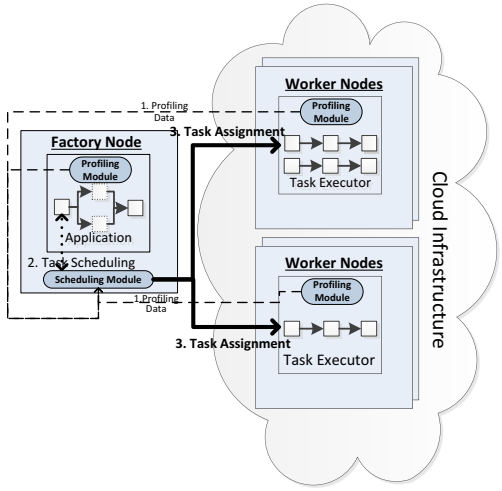


Figure 7: Architecture Overview of the Profiler-based Scaling

Based on the currently available profiling data and the task execution state of each worker, the Scaling System can construct resource usage predictions for each worker. These predictions play a key role in the process of scheduling new tasks. Every time one or more new tasks need to be scheduled, the Scaling System constructs the current prediction for each worker and tries to schedule each new task to start as soon as possible. Generally, the scheduling problem is isomorphic to the well-studied bin-packing problem, which is known to be an NP-hard problem [32]. Hence, our scheduling approach is currently based on a heuristic greedy algorithm. We consider other implementations, for instance, based on evolutionary algorithms [33] in our future research.

In order to formally define our scheduling goal, we need to define some additional preliminaries. The algorithm schedules instances of different types of tasks ($\tau_n \in T_i$), where each type of task T_i has known or previously measured expected resource usage profiles for each profiled resource $\rho \in P$ ($\forall \rho \in P : I_\rho$), as explained in Section IV-C. Additionally, each type of the tasks has an expected duration (t_{T_i}), which is the minimal constraint for each task instance execution time ($t_{\tau_i} \geq t_{T_i}$). The expected task execution time (t_{τ_i}) is determined during the scheduling process and is caused by delays because of inter-task competition over computational resources or deliberate task suspension. Each profiled resource ρ_x on each cloud host $h_i \in H$ has an usage limitation ρ'_x, after which either task execution slows down due to competition with concurrent tasks for competitive resources, or the application runs out of available resources and crashes for cumulative resources (e.g., with an `OutOfMemoryError` for Java applications). Additionally, if some obtained cloud resource is not utilized above some boundary value ρ_{b_x}, the corresponding host is assigned a penalty $p_{h_i}(\rho_x, \rho_{b_x})$. Based on this formal model, our scheduling approach aims to minimize the total task processing time after the initial startup time t_0, while keeping resource wasting (as captured via penalties) as low as possible (see Equation 3). Effective resource usage and task execution time often represent conflicting choices, hence, application developers can additionally specify which of these criteria is more important via the coefficients (A, B).

$$S = A \sum_{h_i \in H} \Big(\sum_{\rho_x \in P} (p_{h_i}(\rho_x, \rho_{b_x})) \Big) + B(max(t_{\tau_i \in T_i}) - t_0) \rightarrow min \quad (3)$$

The greedy scheduling approach is shown in Figure 8. First, the Scaling System sorts all new tasks in correspondence to their deadline (sooner first) and expected resource usage (larger tasks first). This allows to schedule and run more prioritive and demanding tasks sooner while there are more scheduling options available (less tasks currently running or scheduled). After this ordering, for each new task, an appropriate host and starting time is selected. To do this, the Scheduler tries to determine how soon current task can be started on each available host, while satisfying all defined resource usage constrains. This is done by including the current task into the execution plan of the host and detecting if any resource constraint is violated. If this is the case, the Scheduler tries to postpone the task further by moving only the first point when the resource constraint was hit. If the required delay for this point is found, a new schedule is calculated using the newly shifted startup time. Note that the scheduling algorithm cannot use the current time as task startup time, as it has to postpone the current task for at least the amount of time required to transfer the task over the network to the worker and start the execution there. This time is a parameter of our approach, and can be either measured on startup (e.g., via the round-trip of packet of appropriate size plus task initialization time), detected by observing previous scheduling results, or preconfigured by the application developer. Additionally, if the

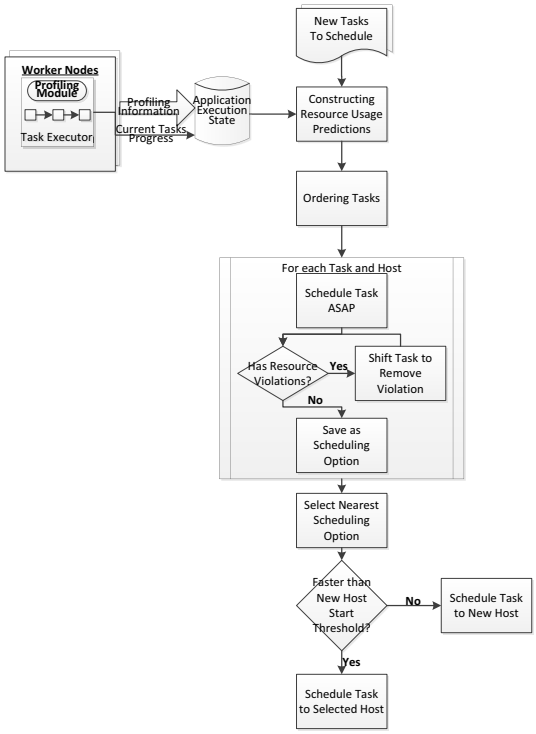


Figure 8: Overview over the task scheduling heuristic

task arrives to the host later than it was scheduled, the task is returned back to the Scheduler for re-scheduling. After a worker and task startup time is defined by the heuristic, the task is sent directly to the worker for execution to not miss the scheduled start time.

One additional scheduling technique available to the Scheduler if active profiling is used, is task suspension. The task that is being scheduled can be suspended at developer-defined points of execution to allow other tasks to pass their resource usage pikes, therefore allowing to fit task execution within the resource constraint even in situations when a non-suspending Scheduler would need to postpone the task startup time after a resource usage peak.

F. Task Scheduling and Cloud Elasticity

If the task execution plan is filled for a long time ahead (longer than the startup time of a new virtual machine in the cloud), and there are more tasks to schedule, the Scheduler starts a new machine and assigns tasks to the new worker. Additionally, new hosts can be configured to start sooner than existing capacity limits are reached. This may increase task execution performance, but may cause hosts to be under-utilized for some time if there will not be enough tasks to load all available resources. To allow scaling down in situations when there is no sufficient load coming to fill running workers, the Scheduler has a set of penalties for not utilized resources on each host. Whenever the currently reported host usage is lower than a defined boundary (e.g., 50%), the host is getting a penalty score that is considered during scheduling. Hosts with lower penalty are considered first, even if the task can be scheduled slightly later on them (the cost of the penalty in comparison to scheduling delay is configured on startup). This allows to free unnecessary workers and shutdown them if there is insufficient load to use effectively all available hosts.

V. Evaluation

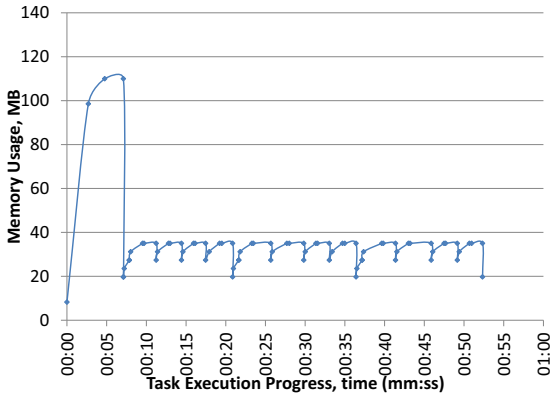


Figure 9: Single Task Execution Memory Profile

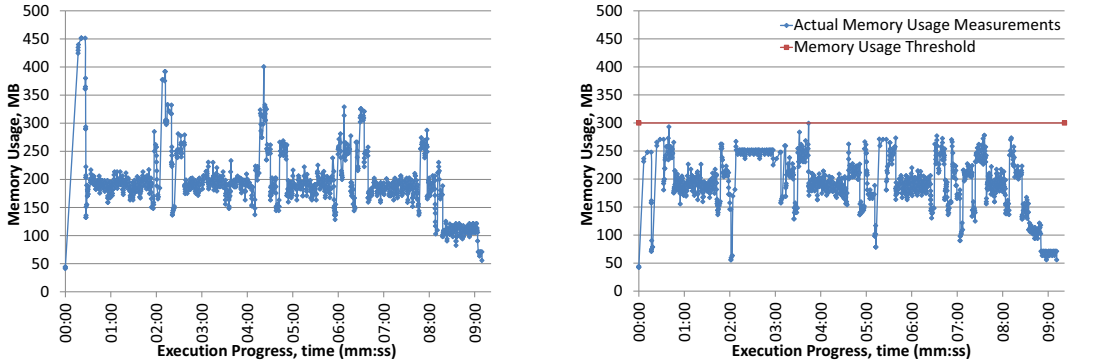
We evaluate the main claims of the paper based on an implementation of the illustrative example sketched in Section II, which shows how memory usage profile can be altered using active profiling. The main goal of our evaluation is to demonstrate how the approach presented in this paper may be used for optimizing resource usage in cloud computing.

A. Evaluation Setup

Our evaluation application is based on the motivation example presented in Section II. It creates a set of image-processing tasks based on the actual collision data, captured by the ALICE detector[1]. As each task applies the same image processing algorithms to the different images of the same size, each task resource usage follows the same execution profile, as shown in Figure 9. This conforms with our algorithm restrictions and allows us to use the presented approach to control application execution. In order to perform a fair comparison and clearly show the impact of active scheduling, all tasks are executed twice: once without any execution control and once with active execution control, based on previously calculated resource usage profiles. In our evaluation application, we focus on memory usage as computational resource. This allows us to simplify the demonstration application and to achieve better reproducibility of results. Thus, the main goal of our evaluation is to show that active profiling allows keeping resource usage under specified boundaries, while not significantly reducing application performance.

We developed an image processing application implemented in C++, which uses the CImg[2] open-source image processing library. It consists of (1) a scheduling system, that has to generate and schedule tasks to execute, (2) a profiling system, that profiles task execution process, reports memory usage to the Scheduler and controls task overlapping in order

[1] http://cds.cern.ch/record/1477949
[2] http://cimg.sourceforge.net



(a) Full application execution without any means of resource usage control

(b) Memory profile captured using the prototype scheduling system

Figure 10: Evaluation Results

to follow the task execution plan and (3) an Executor, which executes tasks and queries the profiler during execution in order to perform memory usage measurement and to ensure scheduled task execution sequence. The execution process is compared to a simple baseline scheduling approach, which does not consider resource profiles or current usage at all, and only schedules based on the amount of cores available on a worker. The full application source code along with all raw evaluation results are available online[3].

B. Evaluation Results

From the single task execution profile (Figure 9), it is clearly visible that the task contains a single significant memory usage spike at the beginning of each execution. This spike is caused by the need to open a large image file in order to determine and select the area interesting for further processing. As it was said before, the execution of such tasks in a cloud environment requires from developers to reserve at least the amount of memory equal to the sum of worst-possible scenarios of possible concurrent task executions. In our case, developers have to reserve at least 120MB of RAM for each concurrent execution. This hypothesis is validated by our baseline task execution, during which all tasks were concurrently executed in 4 available threads without any execution control. The results are shown in Figure 10a. This figure demonstrates that while generally the application requires only about 200MB of RAM, at times of overlapping spikes we require between 300MB to 450MB. Thus, while we have to reserve every time at least 500MB of RAM for such execution schedule, most of the time these resources are not actually used.

In order to improve this situation, we run the same tasks controlled by a prototype of the Scheduler presented in this paper, which was configured to keep memory usage below 300MB. Execution results are shown in Figure 10b. As can be seen, the Scheduler produced a memory usage that is much more homogeneous and constant. This allows us reserving less resources or permitting more parallel task executions, depending on the resources available and application goals. Additionally, the application execution time did not increase significantly. While our scheduled execution used 33% less memory, it only took 1% more time to complete the same execution sequence. Therefore, we can state that this shows that the performance penalty of such task execution control itself is insignificant. Note, that it is important to understand that, in general case, the overall performance effect heavily depends on the executed task profile, the amount of high resource usage spikes and the configured resource usage boundaries.

[3]http://www.infosys.tuwien.ac.at/staff/phdschool/rstzab/papers/2014_EUROMICRO.zip

VI. Conclusions and Future Work

While cloud computing brings the ability to acquire and release resources according to application needs, common resource usage patterns may lead to resource over-provisioning and wastage. In this paper, we focused on a specific subset of cloud computing applications that consist of known sets of uniform tasks with non-uniform resource usage patterns. In order to optimize resource usage, we provided a task scheduling and execution approach based on task execution profiles and resource usage restrictions, defined by application developers. We introduced the core components and concepts of profile-based scheduling and evaluated our approach on an illustrative example application. Our results indicate that the presented approach allows controlling the resource usage of an application, while not influencing drastically the overall performance in the context of the example application.

In our future work, we plan to continue working on profile-based task scheduling in order to decrease usage complexity and internal knowledge required. Particularly, we plan to explore abilities to automatically consider costs and penalties of resource usage violations in comparison to task deadline violations. Additionally, we consider the usage of aspect-oriented programming in order to simplify task profiling and execution control. Furthermore, we plan to investigate alternative scheduling approaches that might increase scheduling performance, while slightly loosing guarantees of the best possible scheduling plan, what is often desirable in real-world applications. Finally, we plan to integrate the presented

scheduling approach with our Cloud middleware JCloudScale, as introduced in [34], [35].

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement 610802 (CloudWave). Also this work is partially supported by the European Union within the SIMPLI-CITY FP7-ICT project (Grant agreement no. 318201).

REFERENCES

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Fut. Gener. Comp. Syst.*, vol. 25, no. 6, pp. 599–616, Jun. 2009.

[2] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *IEEE Fifth International Conference on Cloud Computing (CLOUD 2012)*. IEEE, 2012, pp. 423–430.

[3] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 6, pp. 931–945, Jun. 2011.

[4] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis, "Efficient resource provisioning in compute clouds via vm multiplexing," in *7th international conference on Autonomic computing*. ACM, 2010, pp. 11–20.

[5] F. Buschmann, K. Henney, and D. C. Schmidt, *Pattern-Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing*. Wiley, 2007.

[6] "Computerworld: Collider test called a 'great milestone of mankind'," http://www.computerworlduk.com/news/infrastructure/10948/collider-test-called-a-great-milestone-of-mankind/.

[7] "Attacking cern's big data problem," http://gigaom.com/2013/09/18/attacking-cerns-big-data-problem/.

[8] M. Xu, L. Cui, H. Wang, and Y. Bi, "A multiple qos constrained scheduling strategy of multiple workflows for cloud computing," in *Parallel and Distributed Processing with Applications, 2009 IEEE International Symposium on*. IEEE, 2009, pp. 629–634.

[9] H. Qi-yi and H. Ting-lei, "An optimistic job scheduling strategy based on qos for cloud computing," in *2010 Inter. Conference on Intelligent Comput. and Integrated Systems*. IEEE, 2010, pp. 673–675.

[10] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities," in *International Conference on High Performance Computing & Simulation*. IEEE, 2009, pp. 1–11.

[11] K. Liu, H. Jin, J. Chen, X. Liu, D. Yuan, and Y. Yang, "A compromised-time-cost scheduling algorithm in swindew-c for instance-intensive cost-constrained workflows on a cloud computing platform," *International Journal of High Performance Computing Applications*, vol. 24, no. 4, pp. 445–456, 2010.

[12] P. Leitner, W. Hummer, B. Satzger, C. Inzinger, and S. Dustdar, "Cost-Efficient and Application SLA-Aware Client Side Request Scheduling in an Infrastructure-as-a-Service Cloud," in *5th IEEE International Conference on Cloud Computing*, 2012, pp. 213–220.

[13] P. Hoenisch, S. Schulte, and S. Dustdar, "Workflow Scheduling and Resource Allocation for Cloud-based Execution of Elastic Processes," in *6th IEEE International Conference on Service Oriented Computing and Applications (SOCA 2013)*. IEEE, 2013, pp. 1–8.

[14] S. Schulte, D. Schuller, P. Hoenisch, U. Lampe, S. Dustdar, and R. Steinmetz, "Cost-Driven Optimization of Cloud Resource Allocation for Elastic Processes," *International Journal of Cloud Computing*, vol. 1, no. 2, pp. 1–14, 2013.

[15] U. Lampe, T. Mayer, J. Hiemer, D. Schuller, and R. Steinmetz, "Enabling Cost-Efficient Software Service Distribution in Infrastructure Clouds at Run Time," in *4th IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2011)*. IEEE, 2011.

[16] R. Buyya, S. K. Garg, and R. N. Calheiros, "Sla-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions," in *Cloud and Service Computing (CSC), 2011 International Conference on*. IEEE, 2011, pp. 1–10.

[17] L. Wu, S. K. Garg, and R. Buyya, "Sla-based resource allocation for software as a service provider (saas) in cloud computing environments," in *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*. IEEE, 2011, pp. 195–204.

[18] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Fut. Gener. Comp. Sys.*, vol. 28, no. 5, pp. 755–768, 2012.

[19] R. Buyya, A. Beloglazov, and J. Abawajy, "Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges," *arXiv preprint arXiv:1006.0308*, 2010.

[20] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M. Q. Dang, and K. Pentikousis, "Energy-efficient cloud computing," *The Computer Journal*, vol. 53, no. 7, pp. 1045–1051, 2010.

[21] J. Baliga, R. W. Ayre, K. Hinton, and R. Tucker, "Green cloud computing: Balancing energy in processing, storage, and transport," *Proceedings of the IEEE*, vol. 99, no. 1, pp. 149–167, 2011.

[22] Y. C. Lee and A. Y. Zomaya, "Rescheduling for reliable job completion with the support of clouds," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1192–1199, 2010.

[23] P. Latchoumy and P. S. A. Khader, "Survey on fault tolerance in grid computing," *International Journal of Computer Science & Engineering Survey (IJCSES) Vol*, vol. 2, 2011.

[24] G. Suciu, C. Cernat, G. Todoran, V. Suciu, V. Poenaru, T. Militaru, and S. Halunga, "A solution for implementing resilience in open source cloud platforms," in *Communications (COMM), 2012 9th International Conference on*. IEEE, 2012, pp. 335–338.

[25] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*. ACM, 2011, pp. 301–314.

[26] C. Vecchiola, X. Chu, and R. Buyya, "Aneka: a software platform for .net-based cloud computing," *High Speed and Large Scale Scientific Computing*, pp. 267–295, 2009.

[27] Z. Gong, X. Gu, and J. Wilkes, "Press: Predictive elastic resource scaling for cloud systems," in *Network and Service Management (CNSM)*. IEEE, 2010, pp. 9–16.

[28] F.-f. Han, J.-j. Peng, W. Zhang, Q. Li, J.-d. Li, Q.-l. Jiang, and Q. Yuan, "Virtual resource monitoring in cloud computing," *Journal of Shanghai University (English Edition)*, vol. 15, pp. 381–385, 2011.

[29] "Amazon autoscaling," "http://aws.amazon.com/autoscaling/", Last accessed: 2014.01.22.

[30] "Rackspace cloud monitoring," "http://copperegg.com/rackspace/", Last accessed: 2014.01.22.

[31] S.-k. Kwon and J.-h. Noh, "Implementation of monitoring system for cloud computing environments," *International Journal of Modern Engineering Research (IJMER)*, vol. 3, no. 4, pp. 1916–1918, 2013.

[32] S. Martello and P. Toth, *Knapsack problems*. Wiley New York, 1990.

[33] N. Radcliffe and P. Surry, "Formal Memetic Algorithms," *Evolutionary Computing*, vol. 865, pp. 1–16, 1994.

[34] P. Leitner, B. Satzger, W. Hummer, C. Inzinger, and S. Dustdar, "CloudScale: a novel middleware for building transparently scaling cloud applications," in *27th Annual ACM Symposium on Applied Computing (SAC '12)*. New York, NY, USA: ACM, 2012, pp. 434–440.

[35] P. Leitner, C. Inzinger, W. Hummer, B. Satzger, and S. Dustdar, "Application-Level Performance Monitoring of Cloud Services Based on the Complex Event Processing Paradigm," in *5th IEEE International Conference on Service-Oriented Computing and Applications (SOCA'12)*, 2012.