Hotspot Mitigations for the Masses

Fabien Hermenier
Nutanix
fabien.hermenier@nutanix.com

Aditya Ramesh
Nutanix
aramesh@nutanix.com

Abhinay Nagpal
Nutanix
anagpal@nutanix.com

Himanshu Shukla
Nutanix
hshukla@nutanix.com

Ramesh Chandra
Nutanix
ramesh.chandra@gmail.com

ABSTRACT

In an IaaS cloud, the dynamic VM scheduler observes and mitigates resource hotspots to maintain performance in an oversubscribed environment. Most systems are focused on schedulers that fit very large infrastructures, which lead to workload-dependent optimisations, thereby limiting their portability. However, while the number of massive public clouds is very small, there is a countless number of private clouds running very different workloads. In that context, we consider that it is essential to look for schedulers that overcome the workload diversity observed in private clouds to benefit as many use cases as possible.

The Acropolis Dynamic Scheduler (ADS) mitigates hotspots in private clouds managed by the Acropolis Operating System. In this paper, we review the design and implementation of ADS and the major changes we made since 2017 to improve its portability. We rely on thousands of customer cluster traces to illustrate the motivation behind the changes, revisit existing approaches, propose alternatives when needed and qualify their respective benefits. Finally, we discuss the lessons learned from an engineering point of view.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**; • **Software and its engineering → Virtual machines**; **Scheduling**.

KEYWORDS

Cloud Computing, Virtual Machines, Dynamic Scheduling

ACM Reference Format:
Fabien Hermenier, Aditya Ramesh, Abhinay Nagpal, Himanshu Shukla, and Ramesh Chandra. 2019. Hotspot Mitigations for the Masses. In *ACM Symposium on Cloud Computing (SoCC '19), November 20–23, 2019, Santa Cruz, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3357223.3362717

SoCC '19, November 20–23, 2019, Santa Cruz, CA, USA

ACM ISBN 978-1-4503-6973-2/19/11…$15.00
https://doi.org/10.1145/3357223.3362717

1 INTRODUCTION

A key goal of cloud management is to minimise the cost of operating the cloud, and the VM scheduler is an important component that helps achieve this goal. The VM scheduler uses its precise understanding of how resources (*i.e.* CPU, memory, I/O) are consumed by VMs to compute placements that satisfy customer service level agreements while achieving a high consolidation ratio.

Dynamic scheduling is a hot topic since the last decade and numerous solutions have been proposed [5, 19, 23, 43, 47–49]. A dynamic VM scheduler typically reconsiders the VM placement with live migrations [10] to mitigate hotspots. Traditionally, the cloud community focuses on dynamic schedulers for public clouds. The challenges are then to propose a solution that is effective for a *typical* workload running on the *largest possible infrastructure*. This may lead to workload dependent optimisations and limit their usefulness outside the tested scenario. Also, while the number of massive public clouds is very small, 73% of companies use at least one private cloud [37]. Those infrastructures are usually small in terms of nodes and host a large variety of workloads [8]. Therefore, we consider that it is essential to look for schedulers that support workload diversity to benefit as many users as possible.

ADS is the dynamic scheduler running inside the private clouds managed by the Nutanix operating system to mitigate hotspots. As it is not possible to predict the practical workload that is running inside private clouds, ADS is expected to support workload diversity. In this paper, we present the design, the implementation and the enhancements we made to achieve this objective.

This includes a review of existing approaches, their adaptations to make them fit private cloud workloads and the proposal of alternative solutions. The changes are motivated from practical observations and qualified individually by replaying the mitigation requests emitted by 2,668 customer clusters. The benefits are discussed globally but also at the scale of a single cluster to highlight the risk of double-edged changes that may be globally valuable but locally unacceptable. Finally, we review how tackling workload diversity impacted the engineering process of ADS.

The rest of the paper is structured as follows. Section 2 presents a background of private clouds and the hyper-convergence paradigm. Section 3 discusses the initial design and implementation of ADS. Section 4 reviews and qualifies the enhancements made to support workload diversity. Section 5 discusses the lesson learned at an engineering level. Section 6 presents related work and Section 7 concludes our findings.

2 PRIVATE CLOUDS

We present here the design of the Nutanix hyper-converged system that is used to operate private clouds. We then characterise the clusters and the workloads of these private clouds to highlight their differences with regards to public cloud infrastructures and their diversity.

2.1 Hyper-converged infrastructures

Traditionally, private clouds adopted a three-tier architecture where the *compute-tier* that runs VMs is attached to a centralised *storage-tier* via the *network tier*. In a hyper-converged system, the tiers are not separated. Each node of the cluster serves the compute and memory requirements of the VMs running on it, and a part of the storage requirements for all the VMs on the cluster. The local storage of the nodes is pooled together by a distributed storage controller to present a unified storage system to the running VMs [6, 17]. To provide data locality, the controller places data of VMs on the same disks as the node where the VM resides.

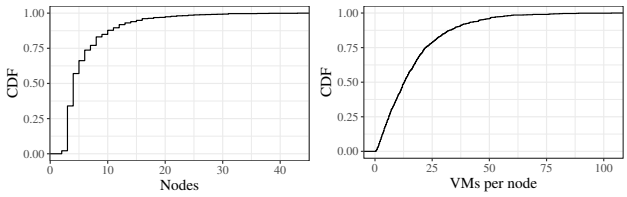
The distributed storage controller consists of one controller VM per node. Each controller manages the local storage on its node and coordinates with the controllers on other nodes to implement storage functions such as data replication, tiering of data across NVMe, SSD, and HDD tiers, and performing optimisations such as compression, deduplication, erasure coding, reading from remote caches, and caching frequently accessed remote data on local hot (i.e NVMe and/or SSD) storage.

The functionalities provided by the controller VMs may be CPU-intensive, especially for random I/O workloads. The performance of an I/O-intensive VM can then be limited by the CPU available to the controller VM serving its I/Os and the contention for CPU among the controller and the user VMs.

2.2 Workload characterisation

Each customer may subscribe to *pulse*, a diagnostic call-home system that shares basic and anonymised system-level information to the support teams. All of the collected calls are stored inside a data warehouse and can be used by engineers to improve the product. We report here a characterisation of the workloads managed by the Nutanix stack using a portion of the *pulse* database. This dataset aggregates 602 customer clusters that used ADS between October 6, 2018 and March 3, 2019. We report here a characterisation of the workloads managed by the Nutanix stack. Overall, these clusters accumulate 5,389 nodes running 101,762 VMs. Each entry in the dataset reports the state of a unique cluster at the moment a hotspot is detected.

2.2.1 Cluster sizing. Figures 1a and 1b show the distributions of cluster size and number of VMs per node, respectively. Figure 1a reports that the clusters tend to be small. The median size is 4 nodes, 12% have 10 nodes or more, while the biggest cluster in the dataset is 43 nodes. Figure 1b shows a median number of 13 VMs per node, 4% having at least 50 VMs per node. Such numbers are explained by the segment occupied by private clouds. Private clouds are single-tenant, and are built to maximise performance per rack using powerful nodes (48 logical cores and 100 GB RAM per node on average for this dataset). With such hardware, small-sized clusters are enough to support the IT requirements of small and medium-sized businesses. Accordingly, private clouds differ fundamentally from public clouds that are multi-tenant infrastructures designed to grab as many tenants as possible, leading to clusters having tens of thousands of nodes, possibly inside a single scheduling space [45].



(a) Nodes per cluster **(b) Consolidation ratio**

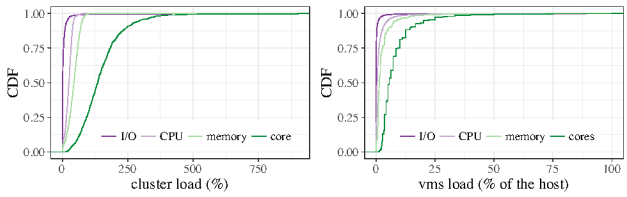
Figure 1: Overview of the cluster size

The *de-facto* standard limit for a single cluster is 64 nodes [15, 46]. When a workload needs more nodes, customers spread it over multiple clusters and a single management pane hides the partitioning to expose a single infrastructure. For example, the biggest public customer running ADS uses more than 2,000 nodes grouped into 32-node clusters. In this context, despite a unique management pane, some services like the dynamic scheduler may be fenced to their respective cluster.

2.2.2 Workload analysis. Figure 2a summarises the distribution of the cluster loads. We first observe that in 75% of the clusters, the CPU cores are oversubscribed. The median load is 135% (*i.e.* 1.35 vCPUs per logical core), 10% of the clusters load their cores to at least 247% with the highest load being 901%. While oversubscribing cores is usually not permitted in public clouds for performance isolation reasons, it is typical in private clouds. In a private cloud, the users control their environment, and thus a best effort CPU allocation based on the VM practical usage is valuable to increase the cluster hosting capacity. We also observe that the memory is more heavily used than CPU and I/O resources. Memory requirements are usually exaggerated as dynamic memory allocation is not encouraged in production clusters while CPU and I/O resources are allocated on demand. Finally, the global load per resource appears moderate: the median CPU and I/O loads for the clusters are 25.25% and 1.38% respectively. This is expected in an enterprise cluster as it is sized to support the simultaneous failure of one or two nodes. A cluster with a few nodes appears then to be oversized with regards to its nominal state.

Figure 2b depicts the distribution of the resource load for the VMs normalised to their hosting node. The load illustrates that a majority of the VMs consume a few resources and a very few VMs consume a significantly large amount of resources, *e.g.* 99% of the VMs consume at most 13% of their host CPU while 0.002% are consuming at least 50%. The very low number of huge VMs is justified by the increase of the node capacity, especially in terms of CPU cores, that exceeds the vast majority of VM requirements.

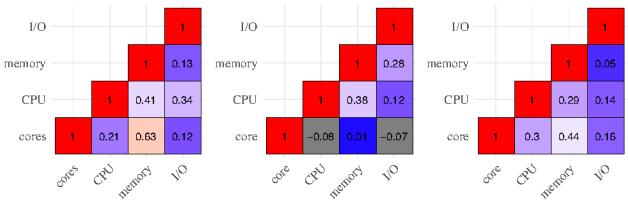
Figures 3a, 3b and 3c depict the Spearman correlation between the resources used by a VM, a node and a cluster. We observe that the correlations between the resources allocated dynamically are



(a) Cluster load (b) VM resource usage

Figure 2: Overview of the cluster load

weak. This highlights a non-homogeneous resource usage, *e.g.* a CPU-heavy VM is not necessarily I/O or memory heavy. Similarly, a node or a cluster may be close to saturation on one dimension while being moderately loaded on another dimension. There is also a weak correlation between the number of cores and the CPU usage for a VM. This reflects that some VMs may be sized with an excessive number of cores compared to their actual usage. We finally observe a correlation between the number of cores and the memory allocated to a VM. This highlights a pattern where VMs are sized homogeneously, a habit possibly inherited from the *de-facto* VM templates of public clouds.



(a) VM consumption (b) Node load (c) Cluster load

Figure 3: Spearman correlation between the resources. Each coefficient is a statistical measure of the strength of a monotonic relationship between two metrics. A value bigger than 0.6 or lower than −0.6 exhibits a strong increasing or decreasing monotonic relationship, respectively.

3 ADS IN A NUTSHELL

This section reviews the features of ADS, its high-level design and implementation.

3.1 Features overview

ADS is a background service that analyses periodically the state of its cluster to identify hotspots. A hotspot indicates that the CPU or the I/O usage of a node is close to its capacity, which can potentially reduce the VMs' performance if there's a sudden spike in resource requirements.

When a hotspot is detected by ADS, it computes a new placement for the VMs that satisfies an estimate of their expected resource usage and honours optional policies such as placement affinities and high-availability expectations. There is no reservation of dedicated cores for VMs as we want to ensure high core over-subscription.

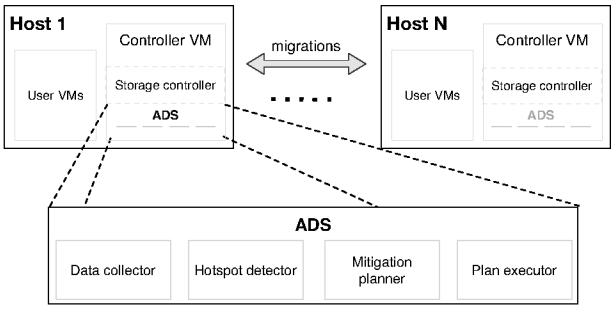


Figure 4: ADS high-level design

The constraint of high-availability states that a cluster is n-FT (Fault Tolerant) if it is possible to immediately reboot the lost VMs when up to any combination of n nodes crash [3, 40]. In practice, this requires saving enough memory on the cluster to restart the lost VMs as CPU and I/O can be overcommitted temporarily.

If a new viable placement exists, the service returns a mitigation plan that indicates which VMs to migrate and the possible dependencies between the migrations. A hotspot may be mitigated by migrating a combination of VMs, but a migration is a costly operation. First, it requires to transfer the VM memory while minimising its downtime [10]. Second, in an hyper-converged environment, it also leads to the transfer of the VM storage working set to reinforce data locality. As a result, a VM migration pressurises the network and affects the storage performance due to the temporary loss of data locality. Accordingly, ADS does not aim at properly balancing the resource usage of the cluster as a re-balancing may involve numerous migrations without necessarily enhancing global performance. Instead, it tends to minimise the data (memory and storage working set) to be transferred during the mitigation.

3.2 Design overview

ADS is a stateless active/passive service deployed on every node of the cluster and runs in the controller VM (CVM). The management operations performed by ADS makes it capable to support any hypervisor supporting live-migration. It is however enabled only on the clusters running AHV, the company hypervisor that is a variant of KVM [29]. As shown in Figure 4, ADS is comprised of four components: a data collector, a hotspot detector, a mitigation planner and a plan executor.

The data collector periodically measures the capacity and utilisation of CPU, I/O and memory resources at the node and the cluster level. It also measures the configured resources and resource utilisation for every VM. The measurements are stored to track historical resource consumption and are used to characterise the workload.

The hotspot detector analyses the workload periodically (by default every 15 minutes) to report any hotspots that currently exist or that may arise in the near future. By default, ADS considers a hotspot on a node if its CPU or its storage controller usage exceeds 85% of the node's capacity. Memory hotspots cannot arise as AHV does not support memory overcommit.

The mitigation planner is invoked if a hotspot is flagged. The planner uses the data collector measurements to compute a mitigation plan if exists. A mitigation plan is a sequence of VM migrations

that removes the hotspot while satisfying the resource demands and placement constraints for all VMs. It is computed using a specialised version of the BtrPlace VM scheduler framework [23].

The plan executor executes the mitigation plan by interfacing with the hypervisors of the nodes to migrate the selected VMs. The migration tasks are done optimistically without blocking existing user operations and can run across crashes.

3.3 Implementation overview

Here, we first present an overview of BtrPlace, the VM scheduling framework that is used as a backend for the mitigation planner. We then discuss how the mitigation service is built on top of it.

3.3.1 BtrPlace basics. BtrPlace is an extensible VM scheduling framework that computes (i) a placement for VMs that satisfies their resource demand, and (ii) a schedule of actions to achieve the new placement [23]. The extensibility is provided by the use of Constraint Programming (CP) [38]. CP solves combinatorial problems, which are modelled by stating constraints (logical relations) that must be satisfied by the solution. The solutions are computed by browsing a search tree built lazily and pruned by the filtering algorithm of each constraint.

BtrPlace placement decisions are customised through *satisfaction-oriented* constraints addressing domain-specific concerns. Also, one *objective* constraint is used to score to each computed solution and tells the CP solver to compute the solution leading to the best score. The optimisation phase is incremental: once a solution is computed, the solver uses its cost as a bound to state a new relation asking for a solution with a better score. This process is repeated until the solver browsed the search tree, proving the solution optimality. Accordingly, if a plan is computed, BtrPlace can prove the solution is the best. If no plan is computed, BtrPlace can prove that the problem is unsolvable or that it was not able to state about the problem feasibility during the allocated time. In the latter case, the problem is considered *undecidable*. This status depicts the *phase transition* in which the probability to find that there is a solution decreases from 1 to 0 as the constraints become increasingly tight.

During the early development stage of the scheduler, different strategies and heuristics were tried but none were generic enough to capture all of the initial use cases. Accordingly, an exact solution appeared desirable to overcome the workload diversity. However, the VM placement problem is NP-hard and the solving time increases exponentially with the number of nodes and VMs. The challenges were then to understand the problem characteristics, model it using appropriate constraints and to implement domain-specific branching strategies to guide the solver efficiently through the search tree, avoiding as much as possible a phase transition for the typical problems we have to solve.

3.3.2 Modelling the desired cluster state.

Modelling hyper-convergence. In a non-hyper-converged system, a CPU-intensive VM and an I/O-intensive VM can be co-located on the same node since they are not contending for the same resource. In a hyper-converged system, by contrast, the storage controller serves I/O requests for the running VMs using the node's CPUs shared with running VMs, which can result in contention between CPU-intensive and I/O-intensive VMs. For example, a CPU-intensive VM reduces the CPU available to the storage controller, which can adversely impact the performance of I/O-intensive VMs on the same node. As a result, a challenge in modelling hyper-convergence is to consider that the host CPU serves both the VM vCPUs and the VM I/Os without any strict reservation.

The data collector generates time-series data by periodically gathering resource usage from different components. It first gathers the CPU and memory capacity of every node. It normalises the CPU measurements to cycles per second to fit hardware heterogeneity. For nodes with SMT, CPU capacity is increased by 10% (instead of 100%) since logical CPUs provide fewer resources than a physical CPU. The 10% value is conservative and was chosen from an analysis of customer workloads.

The data collector also measures the CPU usage of every VM by aggregating the CPU usage of the threads that constitute the VM and the threads virtualising the I/O devices such as storage, network, and console. The memory usage is obtained from its configuration size as the hypervisor does not support memory overcommit.

The data collector measures the I/O load of a VM by measuring the CPU used by the storage controller to serve the VM's I/O requests. For this purpose, the storage controller tracks the time spent to serve the storage request from every VM. The collector couples this information with measurements of the CPU used by the storage controller to approximate the controller's CPU consumption for each storage entity.

The storage controller running on each node may consume CPU for reasons other than serving I/O for VMs, for example, to perform storage garbage collection. This additional CPU usage of the storage controller is considered as a fixed cost since this overhead is incurred regardless of the workload that is running on the cluster.

The time-series data gathered by the collector during the past two hours are used to forecast the VM resource usage and the node capacity for the next 30 minutes using double-exponential smoothing [30]. This approach removes noise and projects a trend.

Modelling hotspot mitigation. The resource usage of VMs running on a contended node does not necessarily match their demand, as the observed values report a lower bound. After a migration, a VM may then start consuming more resources, possibly triggering another hotspot to be mitigated by the next call to ADS. To reduce the probabilities of this hysteresis effect, the values of CPU and storage CPU consumption for a VM running on a contented node are artificially increased, by 5% and 20% respectively. These values were computed empirically—choosing higher values resulted in more unsolvable problems as the resource constraint is more likely to be violated.

3.3.3 Mitigation objective. ADS mitigates observed hotspots using VM migration while minimising the data transfer cost.

The migration cost of a VM consists of two parts : (i) the cost of data transferred over the network as part of the migration process and (ii) the cost of additional data transferred over the network after the migration, due to the impact of migration of storage for maintaining data locality.

The first part of the cost is due to copying of the VM's memory pages from the source node to the target node; this can be done by a pre-copy algorithm [10] where the pages are copied before the

VM is migrated to the target node, or by a post-copy algorithm [25] where the VM is first migrated and the pages are copied later. AHV uses a pre-copy algorithm because it provides both predictable and higher performance after migration, and avoids VM failure if the network between the source and destination nodes fails during the migration. ADS estimates the first cost for a VM migration as the amount of VM memory.

The second part of the cost represents the temporary loss of data locality as the VM's I/O requests go to a new local storage controller that has a lower hit rate in its local data storage. ADS estimates the second cost as the VM's I/Os per second that were satisfied from the hot tier.

The total migration cost of a VM accumulates the normalised memory transfer and data locality costs. Let M and D be the maximum memory size and the maximum data locality of any VM in the cluster, respectively. Let $m(i)$ and $d(i)$ be the memory size and the data locality factor for VM i, respectively. Equation (1) computes the cost $c(i)$ of migrating a VM i. The solver objective is then to minimise the sum of all the migration costs.

$$c(i) = \frac{m(i)}{M} + \frac{d(i)}{D} \quad (1)$$

The branching strategies used to guide the CP solver behind BtrPlace in the search tree helps at minimising the migration and the resource imbalance. A branch in the search tree selects a VM to place and a node to place it on. If a branch leads to violation of a constraint, BtrPlace backtracks and tries another branch. The branching strategies for traditional greedy heuristics involve picking a random node for every VM or involves a worst-fit decrease (WFD) heuristic. The random selection is fast and performs well when there are many VMs that are small compared to the nodes. WFD is efficient when the VMs to place have a single dimension or a uniform load among the dimensions. As discussed earlier, private cloud workloads are non-homogeneous and these strategies led to low-quality solutions in some cases.

The branching strategy of ADS is a variant of WFD. It scores nodes by a triplet consisting of the CPU, memory, and storage-controller CPU load; and scores VMs by a triplet consisting of the CPU, memory, and storage-controller CPU resource consumption. ADS considers the global load of a node (respectively global consumption of a VM), as the highest load (respectively highest consumption) among the three dimensions in the triplet. This reflects that having a node saturated on one dimension has the same consequences on the hosting capacity as being saturated on every dimension. The solver selects VMs in descending order of their normalised global load. By default, it tries to leave a VM on its current node to reduce migration cost. Otherwise, the solver tries the node whose global load with the target VM running on it will be the lowest. The rationale is to co-locate VMs having complementary workloads to reduce the node local imbalance thus maximising its usability (see Figure 5).

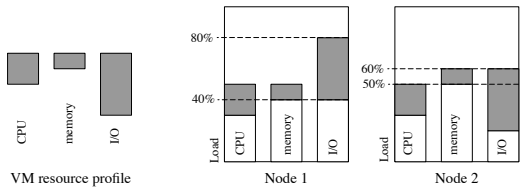


Figure 5: Node selection principle. Node 1 is initially least globally loaded than Node 2 but the latter is a better candidate for the VM to reduce the resulting global load.

4 LESSONS FROM THE WILD

ADS has been enabled in production since November 2016. Over time, we have improved its portability from our understanding of the practical problems that were solved in customer clusters. We discuss here our findings and the enhancements that were made. To qualify the changes, we replay the mitigation requests emitted by customer clusters using different versions of ADS. The dataset of customer calls has been extracted from the pulse knowledge base. It corresponds to calls from 2,668 clusters collected between December 2017 and April 2019.

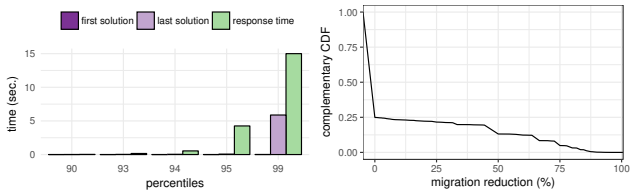
4.1 On the use of an exact approach

We discuss here the benefits of an exact approach to compute and validate the quality of the mitigation plans. Finally, we review the recurring question of the scalability of this approach in the context of private clouds.

4.1.1 The benefits of the optimisation phase. The optimisation phase in ADS is incremental. Each time a solution is computed, it looks for a better solution until a timeout is reached or the entire search tree has been explored. Even if ADS can compute multiple solutions, the practical benefits are not guaranteed: once the first solution is computed, ADS may not be able to compute a better solution within the time limit and a new solution may not reduce the number of migrations significantly.

We review here the practical benefits of the optimisation phase. For evaluation purpose, the first computed solution, the last computed solution and the response time of the service are tracked. The overall response time may differ from the time to compute the last solution as the solver may try to improve the solution without finding one. This situation can happen if it either reaches the solver timeout or proves the optimality of the last solution.

When the solver stops at the first solution, the solver acts as a meta-heuristic as the filtering and the propagation mechanisms explore the search space more efficiently than a greedy heuristic. Within ADS, the branching strategy makes it behave like a Worst Fit heuristic (Section 3.3.3).



(a) solving durations **(b) saved migrations**

Figure 6: ADS **optimisation capabilities**

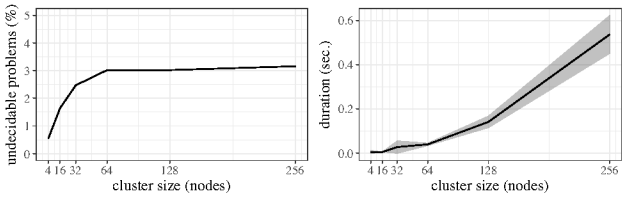
Figure 6a shows the tail percentiles for the different timings of each solved problem. The time to get the first solution is negligible, which is important to ensure a minimum quality of service. The response time differs from the time to the last solution at the 93^{rd} percentile to hit the service timeout from 95^{th} percentile. This indicates that the solver is no longer proving the optimality of the last solution.

Figure 6b reports that the optimisation phase reduces the migrations used to mitigate the hotspots for 25% of the problems and reduces the migrations by half for 13% of the problems. This confirms that the optimisation phase is valuable. Indeed, the savings in terms of migrations is significant while the additional time to compute an improved solution is negligible against the average duration of a migration (30 seconds).

4.1.2 The inherent scalability question. ADS mitigates hotspots using an exact but time-truncated approach. The underlying theoretical problem is NP-hard, and thus ADS, like any exact approach, has scalability limits and the time to solve the problem increases exponentially with the problem size. It matters however to put the limits in perspective to the practical problems to be solved in private clouds and their respective size (Section 2).

We evaluate here the scalability limits of ADS by studying its response time depending on the cluster size. Even though our customer dataset exhibits a large number of small clusters, we know that some of the biggest clusters cannot enable *pulse*. To overcome that bias, we evaluate ADS's limits by scaling up the mitigation requests inside the dataset. Each request from the 768 4-node clusters was duplicated up to reaching 256 nodes and the resulting scaled request was solved using ADS.

This evaluation relies on analysing customer workloads instead of a much larger synthetic workload but since no enterprise cloud providers advise for more than 64 nodes per cluster, it is impossible to analyse real customer traces beyond that scale. We consider however that this approach provides a good insight about what bigger workloads would be. Internally, this approach is used to anticipate hypothetical envisioned workloads.



(a) SLO violations **(b) Mean time to get a first solution and 95% confidence interval**

Figure 7: ADS performance when scaling the clusters

Figure 7a shows the percentage of problems that miss the 30 second response time SLO, reflecting inappropriate performance. The violation rate increases steadily from 0.55% to 3.16%. Figure 7b shows the time to get a first solution for the solved problems. As expected, the solving time increases exponentially. In practice the duration stays negligible compared to the duration of a single VM migration or the service response time. The steady undecidable rate and the short duration prove that the problems that become undecidable are those that were already hard to solve at their original scale, thus quickly putting the solver in phase transition once scaled up.

0.55% of the problems were already undecidable at their original scale. The solver scalability is then not the issue here. 0.96% were initially proved unsolvable, and thus the probability to have it without a solution at a higher scale is high and moving to the undecidable state is not critical. The remaining 1.65% were initially solved and became undecidable at a higher scale. Only those problems reflect the scalability issue. We consider this percentage acceptable as it is related to clusters that are 4 times bigger than the current limits while problem partitioning may be considered as a workaround [23] in case needed.

4.1.3 Conclusion. These experiments prove that an exact but time-truncated approach for a dynamic scheduler is valuable to provide qualitative results against heuristics. More importantly, it is suitable for private clouds in terms of response time and scalability despite this point being frequently discussed.

4.2 Looking for workload agnostic optimisations

Private clouds host a large variety of workloads such as batch processing, virtual desktops, server workloads, databases, *etc.*[8]. This diversity creates a large variance in VM sizing and resource utilisation.

ADS aims at being effective to as many workloads as possible. However, the underlying theoretical problem is NP-hard and some workloads may be easier to address than others. For example, non-uniform VM sizes or heavily loaded clusters may place the solver in phase transition while a model simplification may restrict the usefulness of the service.

We motivate and describe here some optimisations we considered over time and their consequences in terms of solving capabilities or solution quality.

4.2.1 The right local search method. For a placement problem involving n nodes and v VMs, there are n^v possible assignments to check. As discussed in Section 3.3.3, ADS mitigates observed hotspots and does not try to globally balance the cluster load. Furthermore, the workloads in private clouds tend to be steady [8, 44]. It is then unnecessary to reconsider the placement of all the VMs as most of them may already be correctly placed if the hotspot is local.

By default, BtrPlace runs in the *rebuild* mode and considers that every running VM may be relocated. However, BtrPlace proposes also a *repair* mode to speed up the solving process by implementing a local search [22, 23]. This mode approximates the *misplaced* VMs using a heuristic and then generates a CP model where only those VMs can be relocated. The placement problem is combinatorial, hence, it is not possible to select the minimal set of VMs that does not over-filter the search space. Accordingly, if the heuristic underestimates the misplaced VMs, the problem may be flagged incorrectly as unsolvable.

ADS was initially designed around the predominance of local hotspots in mind. Thus, the *repair* mode of BtrPlace was used to

speed up the solving process for large clusters. All of the unit tests and the feedback from the quality assurance team were positive for this approach. However, at the end of 2018, *pulse* flagged problems suspiciously proved unsolvable and the engineers concluded it was over-filtering due to an aggressive problem reduction.

The first improvement consisted in using the *repair* mode initially and to solve the problem again in the *rebuild* mode in case of a proven unsolvable problem. This solving mode is named *repair + rebuild*. Figure 8 shows the percentage of solved problems depending on the solving mode. As expected initially, the *rebuild* mode provides the highest solving capacities. More importantly, this comparison also confirms that the *repair* mode over-filters significantly. Compared to the *rebuild* mode, it prevented to mitigate hotspots for 2.04 percent points of the clusters (54 clusters among the dataset). The *repair + rebuild* mode fixed the over-filtering by increasing the solving capacities by 2.62 percent points compared to *repair*. Unexpectedly, it also outperformed the *rebuild* mode. It appears that the newly solved problems are *undecidable* in the *rebuild* mode but easier to solve in the *repair* mode thanks to the reduction that moved the solver out of phase transition. Overall, the *repair + rebuild* mode exhibits the best results despite solving some problems twice. The first pass in the *repair* mode computes quickly a solution if it exists, countering some phase transitions in the solver. Finally, the switch to the *rebuild* mode counters the over-filtering of *repair*.

The *repair* and the *repair + rebuild* mode lead to plans having 1.62% fewer migrations than in the *rebuild* mode. As the solver considers fewer VMs in the *repair* mode, the number of migrations is already lowered resulting in a smaller search space. Thus, the solver is more efficient at minimising the migrations.

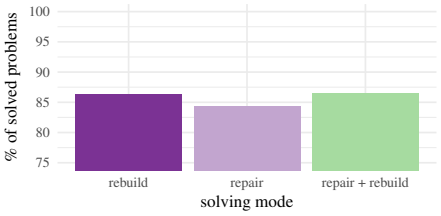


Figure 8: Solved problems depending on the solving mode.

One year after the deployment of *repair + rebuild* feature, two suspicious undecidable problems (not in the present dataset) from customer cases came to the support team. The post-mortem analysis reported that they were undecidable in the *repair* mode but solved in the *rebuild* mode. Unexpectedly, the reduced problems placed the solver in phase transition and became much harder to solve despite being smaller. As a result, the solver wasted all the allotted time in the *repair* phase without having the possibility to switch to the *rebuild* phase. The solving mode was then enhanced to bound the duration of the *repair* phase. The rationale was to identify a solver stuck in the phase transition to switch automatically to the *rebuild* mode, hoping for a possible resolution. The enhancement was considered double-edged as on the current dataset, this improves the number of solved problems by 0.08% but reduces the quality of the mitigation plans by 0.23%. In practice, as ADS should target diverse workloads, the reduction of the solution quality was acceptable. Furthermore, the problem was initially reported by a customer that considered correctly that a solution existed. Accordingly, it was critical to reach this decision capability.

As of today, the revisited *repair + rebuild* solving mode appears satisfactory. However, the use of a timer to catch the phase transition is double-edged. It is always possible to face a workload that can be solved in more than the allotted time for the *repair* phase while being undecidable in the *rebuild* mode. A planned change is to solve the problem in the *repair* and the *rebuild* modes in parallel using a portfolio approach [1]. This removes the need for a mode switch and enables the two solvers to cooperate during the optimisation phase. While this change is easy to perform, it is hard to qualify as ADS runs inside the management VM along with the user VMs on every node. The parallel resolution requires more CPU and memory from the management VM and excessive resource usage may reduce the user VM performance and the overall hosting capacity of every cluster while benefiting a very few use cases.

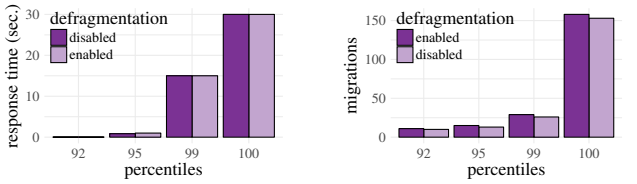
4.2.2 Placement problem or scheduling problem ? A VM scheduler may support cluster defragmentation to increase the hosting capacity. For example, a mitigation service may free up resources from nodes without the hotspot via migrations so that VMs running on the node with the hotspot can migrate to those nodes. This feature requires to consider the VM placement problem as a *Resource-Constraint Project Scheduling Problem* [4] (RCPSP) instead of a *Vector Packing Problem* (VPP) to schedule the migrations and to infer their dependencies.

RCPSP is harder to implement than VPP as it integrates the temporal dimension on top of a spatial problem. It is also harder to solve. Finally, if the scheduler uses a heuristic or a time-bounded exact resolution, the quality of the computed solution may be reduced due to unjustified defragmentations.

An analysis of some hard to solve problems flagged by *pulse* exhibited that the solver was stuck in a region of the search tree devoted to the variables that compute the action schedule. As the mitigation problems exhibit mostly a moderate cluster load (Figure 2a) and local hotspots (Figure 2b), resource fragmentation may not be problematic and the need to support cluster defragmentation may not be valuable.

To assess the practical benefits of supporting cluster defragmentation, the hotspot mitigation problems were solved without the ability to defrag the cluster. When enabled, ADS solves mitigation problems for 2.41% more clusters without creating any undecidable problems. Figure 9a reports that the response time is not globally impacted. Figure 9b reports that this feature does not impact globally the quality of the computed solutions as the size of the mitigation plans seem unchanged. However, despite the global quality staying the same, the solution quality for some problems varies. For less than 1% of the requests, the cluster defragmentation feature reduces the size of the plans by 2 but for 2% of the problems, it increases the plans by at least 75% and up to 96%.

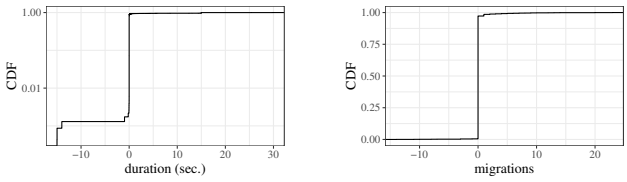
This experiment exhibits that the cluster defragmentation feature is required despite having a few corner cases. While it increases significantly the capacity of ADS at supporting more workloads, it



(a) Response time

(b) Global impact on the mitigation plans

Figure 9: Impact of the defragmentation feature



(a) Response time reduction

(b) Saved migrations

Figure 10: Benefits of the *knapsack* filtering on the solved mitigation requests

reduces the solution quality for a few previously supported workloads. The quality loss is however acceptable as it is limited to some outliers and it does not prevent the service to compute a solution.

4.2.3 Decision capabilities and performance overhead. Mid 2017, a customer case reported *undecidable* mitigation problems for heavily loaded clusters with a few big VMs. Initially, BtrPlace used a specialised version of *vector packing constraint* that was optimised for the synthetic dataset used in their evaluation. This dataset comprised of large problems with thousands of homogeneous nodes but a few, mostly homogeneous, VMs per node. The constraint was specialised for speed and some filtering capacities were removed as they appeared being not valuable.

Section 2.2.2 discussed heterogeneous workloads. Thus, this optimisation inside BtrPlace was not fully appropriate for private cloud. We enhanced the filtering capacities with *knapsack filtering* to handle non-homogeneous workloads. This filtering pro-actively denies VM assignments that will not fit in nodes without having to test and backtrack in case of failure [39]. Knapsack filtering runs in linear time and is effective only when the load on a node is high enough to deny some VMs. For best results, it should be triggered every time the solver places a VM. However, as the clusters are typically lightly loaded, the filtering is not always effective. The algorithm needs also to maintain one ordered list of candidate VMs per dimension for each node, which is costly in terms of CPU and memory resources and despite the first implementation improving the filtering capabilities, the service slowdown was unacceptable. The best compromise between performance overhead and filtering capabilities was achieved with the knapsack filtering enabled when the solver observes that a VM cannot fit on a node and the remaining space is smaller than the biggest VM to place.

We report here the benefits of the knapsack filtering by running the problems with and without the feature. The knapsack filtering reduced the undecidable problems to 0.56% from 0.86% and 50% of the newly solved problems had solutions. While the absolute benefit can be considered small, it enables ADS to address harder problems coming from unsupported workloads. It also contributes to increasing the hosting capabilities and reduce the total cost of ownership for some customers.

Figure 10a reports the response time reduction when the *knapsack* filtering is enabled and Figure 10b reports the number of saved migrations. We observe again that the *knapsack* filtering exhibits corner cases. 0.05% of the problems are solved up to 15 seconds slower and 4% are solved at least 15 seconds faster. 0.05% of the problems led to up to 11 additional migrations while 0.05% saved at least 19 migrations. Again, here, the slowdown and the quality reduction is limited to a very few outliers and more importantly, it does not violate the SLO of ADS.

4.3 Practical effectiveness

The previous evaluations highlighted the theoretical performance of ADS in terms of latency, saved migrations or solving capabilities for customer workloads.

Assessing the practical performance in live conditions is much more complex as *a/b testing* is not possible. Furthermore, with volatile workloads and dynamic allocation of resources, a simple observation of the cluster state before and after a mitigation may be misleading. For example, regardless of what ADS concluded, a hotspot can disappear on its own when VMs are shut down by their users or when the workload running within the VM decreases. The opposite scenario is also possible.

The current method to qualify practical effectiveness of ADS consists in reporting if there is a hotspot after it has applied the mitigation plan. Whenever ADS computes a theoretically viable mitigation plan, there are no more observable hotspots 73.28% of the time once the plan is applied. When ADS states that there is no solution, there are no more observable hotspots only 12.24% of the time once the plan is applied. We then consider that ADS reduces the hotspots by a factor of 6.

We consider that a practical mitigation rate of 73.28% is satisfying, but it matters to discuss why 26.72% of the time ADS still reports at least one hotspot remaining. First, as discussed before, the workload volatility may have created a new hotspot. Second, the performance boost that is used to reduce the hysteresis effect may be too low to prevent future mitigations (see Section 3.3.2) and several mitigation rounds may still be required to overcome consecutive hotspots to reach stability. A bigger boost could speed up the convergence but the risk is to increase the probabilities of creating an unsolvable problem. Finally, using a higher hotspot threshold would also increase the success rate but also miss possible performance issues that could have been mitigated because that threshold was not reached.

For the 12.24% of the hotspots that disappear without intervention of ADS, we consider that a non-zero value is explained by the workload volatility where hotspots may disappear on their own. Furthermore, when ADS states that the hotspot cannot be mitigated, the operator is advised to lower the workload by turning off *low priority* VMs which can result in the removal of the hotspot before the next health check.

5 LESSONS AT THE ENGINEERING LEVEL

We previously motivated and discussed technical changes in ADS since 2017. We discuss here what we learned at an engineering level related to the implementation and maintenance concerns.

5.1 Working with an exact algorithm

5.1.1 The bootstrapping overhead. Developing on top of an exact approach appeared to be harder than using heuristics, especially at the early stage of ADS. Engineers were reluctant to try an exact approach due to the possible response time and CPU/memory usage but proof of concepts removed that concern.

The first issue was the need for theoretical background in combinatorial optimisation, in packing and scheduling problems in particular coupled with the technical expertise of system designs. Hence, it matters to find a compliant solver and to map properly the theoretical model to the library capacity.

When a new feature is developed, the recurring question is about the impact of a change in the model on the implementation scalability, especially in the number of variables and constraints to add. The objective is to control the time spent by ADS in the model construction and the memory consumption of the solver that depends on the number of variables. It matters also to ensure that the modelling constraints fit exactly the practical needs to avoid an over-engineered filtering algorithm that may introduce a costly and unnecessary abstraction layer.

Finally, a lack of optimisation slows down the implementation by orders of magnitude. Thus, the time spent to optimise the model and the code is higher than when using a heuristic. Over the releases, this time decreases as the engineer expertise increases, but the scalability is always considered at the initial stage of any feature design.

5.1.2 Quality self-assessment. Undecidable problems report a lack of filtering capabilities. Their frequency is then a valuable indicator that should be minimised. More importantly, their characterisation help at prioritising the issue and its consequences against other engineering tasks.

Contrary to a heuristic based scheduler, ADS can prove mathematically that the problem has no solution or that the computed solution is the best one according to the objective. This eases the support cases as the engineers spend less time verifying the quality of the computed plans and it is legitimate to argue for a cluster expansion.

5.1.3 Composability helps. Constraints inside a Constraint Programming (CP) solver are independent, their composition is both deterministic and independent from their evaluation order. This improves the scheduler flexibility and testability as each scheduler feature is a plugin implemented using various solver constraints.

Each solver constraint comes with its filtering algorithm to prune the search space. This brings valuable flexibility compared to other exact approaches like mathematical programming. We use *ad-hoc* constraints and filtering algorithms when the standard algorithms are too generic and costly with regards to their practical use or when it is not possible to exactly implement some features. Typically, high-availability (see Section 3.1) is not implemented using the CP model presented in [3] for scalability reasons but through a faster heuristic embedded in a solver constraint and offering the same composability.

5.2 It is all about the resource modelling

5.2.1 Validating a model accuracy is hard. Metrics like the response time or provability qualify the implementation of ADS but cannot validate the accuracy of the theoretical model. For example, the proved unsolvable status is double-edged: despite the solver proving that it is impossible to satisfy simultaneously all the constraints, the assumption is theoretical and solely related to the model that defines an acceptable state for a cluster, especially with regards to resource allocation. Accordingly, as the CPU and the storage-controller resource allocation is dynamic and based on performance counters, a pure resource assignment problem that is proved unsolvable may highlight an over-conservative allocation model. Besides, when the model is modified, it is challenging to predict how the change will affect the practical quality of ADS without running the system using predefined, possibly biased, benchmarks.

The use of simulators like SimGrid [9, 26] have already been discussed as a possibility to replay customer environments and to ease qualifying the theoretical model. However, the simulator itself must provide a model for hyper-converged environments and to the best of our knowledge, no such simulator exists, possibly due to the complexity of the task or the lack of reference environments. In this circumstance, the challenge stays open as the modelling effort and its qualification must be conducted first at the simulator level, before being done at the scheduler level.

5.2.2 Thresholds, weighting functions: a love-hate relationship. ADS is configured by some threshold values and weighting functions, for example to state if a node has a hotspot or to measure the performance boost induced by SMT. Threshold-based reasoning helps the users at understanding what is happening when they are related to observable metrics. At the engineering level, the use of thresholds and weighting functions to calibrate the scheduler is also convenient in the short term when it becomes tedious to model objectively the interactions between the resources or to reconcile different concerns. It finally eases testing if the thresholds are easy to reproduce.

The recurring issue with model parameters is the impossibility to find the values that fit all use cases. For example, some CPU intensive workloads may exhibit performance issues well before reaching the mitigation trigger point. The default values are defined empirically from internal testbeds running different benchmarks and when a range of values is possible, a conservative value is picked to prevent an aggressive simplification. It is always possible to change the configuration parameter without having to restart ADS. This operation is performed manually by the support team but customers are reluctant to move away from the default values as they expect them to be workload agnostic.

Long term, we consider that it matters to reduce such configuration parameters, first to minimise the calibration issues, second because their coordination may lead to undesirable effects. However private clouds allocate resources dynamically from low-level performance counters and the complexity of the current hardware make them unpredictable and complicates the definition of an accurate and scalable scheduler model.

The second approach is to look for learning methods to fit autonomously the values to the workload. This approach is highly desirable and has already been used successfully [28]. *A/b testing* is however not possible in private clouds. Furthermore, populating a training set per cluster to highlight its uniqueness may take too much time given the size of the clusters, while the training algorithms may be too resource intensive to run along with the user VMs without impacting its hosting capacity.

5.3 Observations over assumptions

As discussed in Section 4.2, an enhancement may be double-edged and the corner cases are complex to identify. A missed regression may lead to numerous support cases that will impact customer satisfaction or the engineer's productivity for a long period. Accordingly, while it always matters to support more workloads, the primary concern is to avoid performance regressions.

The work done on the scheduler is then as important as the work done with pulse. The internal enhancements of ADS are driven from an analysis of its behaviour in customer and internal clusters. This approach appeared more rewarding than the traditional code review and testing systems to exhibit issues due to workload diversity.

A key development objective is to always work based on practical data instead of assumptions. When a new feature is planned, one of the first tasks is to verify if assumptions hold using pulse and to deploy the missing data collectors if needed. This helps in validating the potential benefits of the feature early in the development stage and to validate with high confidence before the release.

The size and diversity of the dataset are also crucial to avoid any bias. As discussed before, it improves the identification of double-edged changes before the release. Furthermore, it increases the engineer expertise about what is happening in practice on clusters instead of relying on assumptions that misestimate reality.

One downside however, is that it tends to lower the absolute impact of the improvements as over time, the changes target mostly the outliers. For example, even though the improved filtering discussed in 4.2.3 was effective for only 0.4% of the customer clusters, it plays a significant role to support all the workloads mixing very big and very small VMs.

5.4 The scheduler ecosystem

The scheduling problems are hard to solve manually because of the number of parameters and the number of entities involved. It matters thus to provide tools to help the engineering teams at debugging ADS. As of today, we estimate that half the engineering time related to the scheduler development is spent on debugging tools.

A dashboard reports daily performance indicators from the supervised clusters. This accounts for the response status, timing and the plan quality associated to every call of ADS. Furthermore, scripts extract *suspicious* requests. Typically, these are the undecidable problems, the proved unsolvable problems and the mitigation plans with a high number of migrations.

A significant number of customer cases are related to VMs that seem to be migrated too often or when the notion of a hotspot is unclear. The support team relies on a log analyser to retrace a cluster load and the VM position over time to clarify any over-reaction. A Web service also visualises a cluster state and can re-execute any mitigation request using the last production code. This appeared to be very helpful to instantaneously check if an observable issue was already fixed.

Finally, ADS has a dedicated performance regression tool. The test suite is populated by the mitigation requests in pulse and the support tickets. It identifies the code changes that lead to performance regression or improvements as the coverage of the legacy tools (unit and functional tests) is not sufficient. The test suite also contains scaled-up mitigation requests (see Section 4.1.2) to anticipate possible scalability limits.

6 RELATED WORK

Workload characterisation of production systems. Cano *et al.*[8] already characterised enterprise clouds managed by the Nutanix stack in terms of infrastructure sizing and durability. Cortez *et al.* [11] characterised the workload of Microsoft Azure, focusing on the VM sizing and the submission queue. Mishra *et al.* [32] and Liu *et al.* [31] characterised a Google cluster workload. Finally, Amvrosiadis *et al.* [2] discussed the notion of workload heterogeneity and the importance of supporting diversity in job schedulers. We discussed a workload characterisation but we focused on thousands of private clouds instead of one massive public cloud or clusters devoted to job scheduling. We also highlighted the notion of workload diversity but in the context of private clouds. More importantly, we discussed what we have done at a production level with ADS to overcome that diversity.

Job schedulers. Recent job schedulers explored scalable designs, low latencies and efficient balancing [7, 11–14, 16, 18, 27, 34, 42, 45]. These works focused on computing the initial placement of tasks or VMs depending on their description or past executions while we discuss a dynamic scheduler for hyper-converged clouds that mitigate hotspots from an observation of the workload. More importantly, we highlight the challenge of being valuable to as many workloads as possible instead of supporting particular workloads the best way possible. [7, 12–14, 18, 27, 34, 42, 45] addressed data-locality as a part of a weighting function similar to our migration cost. Cortez *et al.* [11] considered a remote centralised storage and Medea [16] stated VM-VM anti-affinity was enough to address I/O interference.

Dynamic scheduling. Past research on dynamic scheduling focused on three-tier architectures where compute nodes are separate from the storage cluster. The seminal papers [5, 43, 48] focused on dynamic consolidation of the compute tiers by only accounting for the CPU and memory usage. Entropy [24] addressed the migration cost and their dependencies and the follow-up BtrPlace [23] focused on the scheduler flexibility. Xioa *et al.* [49] considered CPU, memory, and network resource usage to model the multi-dimensional resource demands of VMs and then do rebalancing of VMs across servers. A-DRM [47] performed load balancing while considering the memory interference between the VMs. Gulati *et al.* [19] reviewed the VMWare dynamic scheduler. They presented the design, the implementation and the lessons learned. The validation relied

on a simulation and an internal testbed executing proof of concept experiments. We review ADS from customer clusters and the enhancements we made to overcome workload diversity.

For the storage tiers, virtual disks are balanced inside a centralised SAN to avoid hotspots. Basil *et al.* [20] built performance models of storage devices to place or migrate the different virtual disks on different LUNs. Pesto [21] leveraged [20] with a workload injector instead of relying on passive observations of the storage devices. Romano [36] also focused on building storage models with heterogeneous hardware in responding to different workloads, and workload interference on storage systems.

ADS was inspired by all of these works and especially BtrPlace as it relies on its flexibility. However, all the above systems have been built to rebalance either the compute part or the storage part of the VMs inside three-tiers architecture where the compute and the storage tiers are independent. To the best of our knowledge, the current paper is the first to present a production-grade dynamic scheduler that considers the interplay between storage and CPU utilisation.

Resource interference management. Like ADS, other systems explored the problem of modelling dependencies and interferences between resources. [12–14, 33, 50] focused on the detection and the resolution of interferences between VMs or tasks consuming shared resources, typically processor caches or memory buses. They however did not consider the possible interferences with a local storage controller. Singh *et al.* [41] monitored end-to-end application resource usage on compute nodes, network switches, and storage nodes to model the problem of migrating VMs and virtual disks as a multi-dimensional knapsack problem. Although Harmony considers CPU and storage, it assumes however, that the compute and storage clusters are independent of each other, and considers VM and virtual disk migrations as independent actions. AutoControl [35] dynamically modifies the CPU and I/O bandwidth allocation of VMs to maintain application performance. However it does not consider VM migrations to fix bottlenecks.

7 CONCLUSION

We presented ADS, the dynamic scheduler used inside private clouds managed by Nutanix to mitigate hotspots inside thousands of clusters. We discussed its design and its implementation. More importantly, we reviewed how we handle workload diversity through a qualitative analysis of significant changes we made since 2017.

The use of an exact solver slows down feature development but provides valuable benefits in terms of solving capabilities, quality self-assessment and extensibility while not causing any scalability issues contrary to common belief. To the best of our knowledge, the theoretical foundations of the placement problems prevent to have a unique solution to completely overcome workload diversity. The usefulness is improved over time through different optimisations targeting specific outliers. And while some improvements are safe, others may be double-edged, requiring to establish a trade-off between solving capabilities, response time and solution quality. Tools are then crucial to exhibit the outliers, to understand their particularities and to rely on an up-to-date knowledge base to test the changes and identify any potential regressions.

ACKNOWLEDGMENTS

Many people contributed to the work described in this paper. Members of the ADS team who contributed to this design and its implementation include Srinivas Bandi, Igor Grobman, Rahul Singh and Hexin Wang. The final version was much improved by comments from the anonymous reviewers and our shepherd.

AVAILABILITY

ADS develops extensions on top of BtrPlace that are specific to its hyper-converged environments. However all the bug fixes and the enhancements that are not tight to the Nutanix environments are backported to BtrPlace to also benefit to the open-source community.

REFERENCES

[1] Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. 2015. A Multicore Tool for Constraint Solving. In *Proceedings of the 24th International Conference on Artificial Intelligence*. AAAI Press, 232–238. http://dl.acm.org/citation.cfm?id=2832249.2832281

[2] George Amvrosiadis, Jun Woo Park, Gregory R. Ganger, Garth A. Gibson, Elisabeth Baseman, and Nathan DeBardeleben. 2018. On the Diversity of Cluster Workloads and Its Impact on Research Results. In *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference*. USENIX Association, Berkeley, CA, USA, 533–546. http://dl.acm.org/citation.cfm?id=3277355.3277407

[3] Eyal Bin, Ofer Biran, Odellia Boni, Erez Hadad, Eliot K. Kolodner, Yosef Moatti, and Dean H. Lorenz. 2011. Guaranteeing High Availability Goals for Virtual Machine Placement. In *Proceedings of the 2011 31st International Conference on Distributed Computing Systems*. IEEE Computer Society, Washington, DC, USA, 700–709. https://doi.org/10.1109/ICDCS.2011.72

[4] J. Blazewicz, J.K. Lenstra, and A.H.G.Rinnooy Kan. 1983. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics* 5, 1 (1983), 11 – 24. https://doi.org/10.1016/0166-218X(83)90012-4

[5] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. 2007. Dynamic Placement of Virtual Machines for Managing SLA Violations. In *2007 10th IFIP/IEEE International Symposium on Integrated Network Management*. IEEE, 119–128.

[6] Dhruba Borthakur et al. 2008. HDFS architecture guide. *Hadoop Apache Project* 53 (2008).

[7] Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou. 2014. Apollo: Scalable and Coordinated Scheduling for Cloud-scale Computing. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*. USENIX Association, Berkeley, CA, USA, 285–300. http://dl.acm.org/citation.cfm?id=2685048.2685071

[8] Ignacio Cano, Srinivas Aiyar, and Arvind Krishnamurthy. 2016. Characterizing Private Clouds: A Large-Scale Empirical Analysis of Enterprise Clusters. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*. ACM, New York, NY, USA, 29–41. https://doi.org/10.1145/2987550.2987584

[9] Henri Casanova, Arnaud Legrand, and Martin Quinson. 2008. SimGrid: A Generic Framework for Large-Scale Distributed Experiments. In *Proceedings of the Tenth International Conference on Computer Modeling and Simulation*. IEEE Computer Society, Washington, DC, USA, 126–131. https://doi.org/10.1109/UKSIM.2008.28

[10] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. 2005. Live Migration of Virtual Machines. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*. USENIX Association, Berkeley, CA, USA, 273–286. http://dl.acm.org/citation.cfm?id=1251203.1251223

[11] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. 2017. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, New York, NY, USA, 153–167. https://doi.org/10.1145/3132747.3132772

[12] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-aware Scheduling for Heterogeneous Datacenters. *SIGPLAN Not.* 48, 4 (March 2013), 77–88. https://doi.org/10.1145/2499368.2451125

[13] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-efficient and QoS-aware Cluster Management. *SIGPLAN Not.* 49, 4 (Feb. 2014), 127–144. https://doi.org/10.1145/2644865.2541941

[14] Christina Delimitrou, Daniel Sanchez, and Christos Kozyrakis. 2015. Tarcil: Reconciling Scheduling Speed and Quality in Large Shared Clusters. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*. ACM, New York, NY, USA, 97–110. https://doi.org/10.1145/2806777.2806779

[15] Edison Group. 2018. Hyper-Converged Infrastructure Portfolio Comparison. https://www.emc.com/collateral/analyst-reports/edison-dellemc-hci-competitive.pdf.

[16] Panagiotis Garefalakis, Konstantinos Karanasos, Peter Pietzuch, Arun Suresh, and Sriram Rao. 2018. Medea: Scheduling of Long Running Applications in Shared Production Clusters. In *Proceedings of the Thirteenth EuroSys Conference*. ACM, New York, NY, USA, Article 4, 13 pages. https://doi.org/10.1145/3190508.3190549

[17] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google File System. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*. ACM, New York, NY, USA, 29–43. https://doi.org/10.1145/945445.945450

[18] Ionel Gog, Malte Schwarzkopf, Adam Gleave, Robert N. M. Watson, and Steven Hand. 2016. Firmament: Fast, Centralized Cluster Scheduling at Scale. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. USENIX Association, Berkeley, CA, USA, 99–115. http://dl.acm.org/citation.cfm?id=3026877.3026886

[19] Ajay Gulati, Anne Holler, Minwen Ji, Ganesha Shanmuganathan, Carl Waldspurger, and Xiaoyun Zhu. 2012. Vmware distributed resource management: Design, implementation, and lessons learned. *VMware Technical Journal* 1, 1 (2012), 45–64.

[20] Ajay Gulati, Chethan Kumar, Irfan Ahmad, and Karan Kumar. 2010. BASIL: Automated IO Load Balancing Across Storage Devices.. In *FAST*, Vol. 10. 169–182.

[21] Ajay Gulati, Ganesha Shanmuganathan, Irfan Ahmad, Carl Waldspurger, and Mustafa Uysal. 2011. Pesto: online storage performance management in virtualized datacenters. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 19.

[22] Fabien Hermenier, Sophie Demassey, and Xavier Lorca. 2011. Bin Repacking Scheduling in Virtualized Datacenters. In *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming*. Springer-Verlag, Berlin, Heidelberg, 27–41. http://dl.acm.org/citation.cfm?id=2041160.2041167

[23] Fabien Hermenier, Julia Lawall, and Gilles Muller. 2013. BtrPlace: A Flexible Consolidation Manager for Highly Available Applications. *IEEE Transactions on Dependable Secure Computing* 10, 5 (Sept. 2013), 273–286. https://doi.org/10.1109/TDSC.2013.5

[24] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. 2009. Entropy: a consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. ACM, 41–50.

[25] Michael R. Hines, Umesh Deshpande, and Kartik Gopalan. 2009. Post-copy Live Migration of Virtual Machines. *SIGOPS Oper. Syst. Rev.* 43, 3 (July 2009), 14–26. https://doi.org/10.1145/1618525.1618528

[26] T. Hirofuchi, A. Lebre, and L. Pouilloux. 2018. SimGrid VM: Virtual Machine Support for a Simulation Framework of Distributed Systems. *IEEE Transactions on Cloud Computing* 6, 1 (Jan 2018), 221–234. https://doi.org/10.1109/TCC.2015.2481422

[27] Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew Goldberg. 2009. Quincy: Fair Scheduling for Distributed Computing Clusters. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. ACM, New York, NY, USA, 261–276. https://doi.org/10.1145/1629575.1629601

[28] Changyeon Jo, Youngsu Cho, and Bernhard Egger. 2017. A Machine Learning Approach to Live Migration Modeling. In *Proceedings of the 2017 Symposium on Cloud Computing*. ACM, New York, NY, USA, 351–364. https://doi.org/10.1145/3127479.3129262

[29] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. 2007. KVM: the Linux virtual machine monitor. In *Ottawa Linux Symposium*. 225–230.

[30] Joseph J. LaViola. 2003. Double Exponential Smoothing: An Alternative to Kalman Filter-based Predictive Tracking. In *Proceedings of the Workshop on Virtual Environments 2003*. ACM, New York, NY, USA, 199–206. https://doi.org/10.1145/769953.769976

[31] Zitao Liu and Sangyeun Cho. 2012. Characterizing Machines and Workloads on a Google Cluster. In *Proceedings of the 2012 41st International Conference on Parallel Processing Workshops*. IEEE Computer Society, Washington, DC, USA, 397–403. https://doi.org/10.1109/ICPPW.2012.57

[32] Asit K. Mishra, Joseph L. Hellerstein, Walfredo Cirne, and Chita R. Das. 2010. Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters. *SIGMETRICS Performance Evaluation Review* 37, 4 (March 2010), 34–41. https://doi.org/10.1145/1773394.1773400

[33] Ripal Nathuji, Aman Kansal, and Alireza Ghaffarkhah. 2010. Q-clouds: Managing Performance Interference Effects for QoS-aware Clouds. In *Proceedings of the 5th European Conference on Computer Systems*. ACM, New York, NY, USA, 237–250. https://doi.org/10.1145/1755913.1755938

[34] Kay Ousterhout, Patrick Wendell, Matei Zaharia, and Ion Stoica. 2013. Sparrow: Distributed, Low Latency Scheduling. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, New York, NY, USA, 69–84. https://doi.org/10.1145/2517349.2522716

[35] Pradeep Padala, Kai-Yuan Hou, Kang G Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, and Arif Merchant. 2009. Automated control of multiple virtualized resources. In *Proceedings of the 4th ACM European conference on Computer systems*. ACM, 13–26.

[36] Nohhyun Park, Irfan Ahmad, and David J. Lilja. 2012. Romano: Autonomous Storage Management Using Performance Prediction in Multi-tenant Datacenters. In *Proceedings of the Third ACM Symposium on Cloud Computing*. ACM, New York, NY, USA, Article 21, 14 pages. https://doi.org/10.1145/2391229.2391250

[37] Rightscale. 2019. RightScale 2019 State of the Cloud Report.

[38] Francesca Rossi, Peter van Beek, and Toby Walsh (Eds.). 2006. *Handbook of Constraint Programming*. Elsevier Science Inc., New York, NY, USA.

[39] Paul Shaw. 2004. A Constraint for Bin Packing. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming*. Springer-Verlag, Berlin, Heidelberg, 648–662. https://doi.org/10.1007/978-3-540-30201-8_47

[40] B. Shen, R. Sundaram, A. Russell, S. Aiyar, K. Gupta, A. Nagpal, A. Ramesh, and H. Shukla. 2017. High Availability for VM Placement and a Stochastic Model for Multiple Knapsack. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. 1–9. https://doi.org/10.1109/ICCCN.2017.8038384

[41] Aameek Singh, Madhukar Korupolu, and Dushmanta Mohapatra. 2008. Server-storage virtualization: integration and load balancing in data centers. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press, 53.

[42] Alexey Tumanov, Timothy Zhu, Jun Woo Park, Michael A. Kozuch, Mor Harchol-Balter, and Gregory R. Ganger. 2016. TetriSched: Global Rescheduling with Adaptive Plan-ahead in Dynamic Heterogeneous Clusters. In *Proceedings of the Eleventh European Conference on Computer Systems*. ACM, New York, NY, USA, Article 35, 16 pages. https://doi.org/10.1145/2901318.2901355

[43] Akshat Verma, Puneet Ahuja, and Anindya Neogi. 2008. pMapper: power and migration cost aware application placement in virtualized systems. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*. Springer-Verlag New York, Inc., 243–264.

[44] Akshat Verma, Juhi Bagrodia, and Vimmi Jaiswal. 2014. Virtual Machine Consolidation in the Wild. In *Proceedings of the 15th International Middleware Conference*. ACM, New York, NY, USA, 313–324. https://doi.org/10.1145/2663165.2663316

[45] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale Cluster Management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems*. ACM, New York, NY, USA, Article 18, 17 pages. https://doi.org/10.1145/2741948.2741964

[46] VMWare. 2018. VMWare vSAN 6.6 Technical Overview.

[47] Hui Wang, Canturk Isci, Lavanya Subramanian, Jongmoo Choi, Depei Qian, and Onur Mutlu. 2015. A-DRM: Architecture-aware distributed resource management of virtualized clusters. *ACM SIGPLAN Notices* 50, 7 (2015), 93–106.

[48] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. 2007. Black-box and Gray-box Strategies for Virtual Machine Migration. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation*. USENIX Association, Berkeley, CA, USA, 17–17. http://dl.acm.org/citation.cfm?id=1973430.1973447

[49] Zhen Xiao, Weijia Song, and Qi Chen. 2013. Dynamic resource allocation using virtual machines for cloud computing environment. *IEEE Transactions on parallel and distributed systems* 24, 6 (2013), 1107–1117.

[50] Xiao Zhang, Eric Tune, Robert Hagmann, Rohit Jnagal, Vrigo Gokhale, and John Wilkes. 2013. CPI2: CPU Performance Isolation for Shared Compute Clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, New York, NY, USA, 379–391. https://doi.org/10.1145/2465351.2465388